

BUILDING TREES, HUNTING FOR TREES,  
AND COMPARING TREES  
THEORY AND METHODS IN PHYLOGENETIC ANALYSIS

A THESIS  
SUBMITTED IN PARTIAL FULFILMENT  
OF THE REQUIREMENTS FOR THE DEGREE  
OF  
DOCTOR OF PHILOSOPHY IN MATHEMATICS  
IN THE  
UNIVERSITY OF CANTERBURY  
by  
David Bryant

University of Canterbury  
1997



## Abstract

Phylogenetics is the study and identification of evolutionary patterns and structures in nature; this thesis explores the mathematics of these structures. The basic objects of study are the leaf labelled tree and its substructures: quartets, splits, clusters and rooted triples, among others. We present fundamental theorems and characterisations, as well as efficient algorithms for a range of phylogenetic problems.

It is often possible to deduce phylogenetic information not in the original data. We characterise an intriguing system of inference 'rules' that arise in this way, and prove that there exist rules of every order that cannot be reduced to lower order rules.

We describe a polynomial time algorithm that extracts maximum weight bounded degree trees from a given binary character set. The algorithm enables compatibility analysis of large data sets, in this case the daunting "Out of Africa" human mtDNA sequences. Other applications include consensus, quartet puzzling and split decomposition.

We accelerate the Minimum Evolution method with an optimal  $O(n^2)$  time algorithm for calculating OLS edge lengths and fast algorithms for WLS and GLS edge lengths. We show how a Minimum Evolution tree can be efficiently extracted from a collection of splits.

Consensus methods are surveyed, characterised and classified. A new intuitive consensus method for edge weighted trees is introduced, together with an efficient algorithm for constructing it.

We present an algorithm for the Maximum Agreement Subtree problem that is based on rooted triples and is much simpler than existing algorithms. We also provide algorithms for obtaining agreement subtrees with the largest number of edges, rooted triples or quartets.

Issues of complexity are discussed throughout the thesis, with several new NP-completeness results and a list of standard NP-complete phylogenetic problems.



Most of Chapter 3 and parts of Chapter 2 appeared in the paper “Extension operations on sets of leaf labelled trees” written jointly by Dr Mike Steel and me. The paper was published in *Advances in Applied Mathematics*, volume 16, 1995 (pages 425-453).

The Hunting for Trees algorithms in Chapter 4, as well as the human mtDNA analysis appeared in the paper “Hunting for trees in binary character sets: efficient algorithms for extraction, enumeration, and optimization” written by me. The paper was published in the *Journal of Computational Biology*, volume 3, number 2, 1996 (pages 275-288).

The remainder of this thesis is, unless otherwise indicated, my own work. In accordance with mathematics protocol I will endeavour to use the personal pronoun ‘we’ to indicate the reader and the writer, rather than the ‘royal we’.

David Bryant, March 13, 1997.



## Acknowledgements

First and foremost, I wish to thank my supervisor Dr Mike Steel. Mike has always been friendly and encouraging, maintaining the delicate balance between support and challenge. He is an inspiring mathematician and most importantly, a nice guy too.

This thesis was completed with the help of a Canterbury Doctoral Scholarship. Thank-you also to the Mathematics and Statistics department, the New Zealand Mathematics Society and the Universität Bielefeld for travel subsidies and financial assistance.

Thanks to all those people in the various communities I have been a part of during my Ph.D. study:

- to the evening worship group at St Ninians, for a Christian spirituality that fits
- to my extended family at St Tims, for fun, friendship and support
- to the Student Christian Movement Waitaha. Without SCM I may have got this thesis done in half the time—but I'd also be severely deprived. Big warm fuzzy thanks.

Thanks to all of those friends that have made my life liveable, especially:

- to my comrades in Maths, Chris and Burkard
- to Les Brighton, for keeping me sane, and to Angela
- to Alison Paulin, for encouragement at the right time
- to Phill, Andy and Tim, for being great friends with a sympathetic ear
- to Anna, Steph and Julia, for helping me through some hard times
- to my flatmates, past and present
- to Mary and Fiona, for letters and distractions
- to Alison Greenaway, for inspiration
- to  $Li_2CO_3$ , for doing what its supposed to.

Finally I want to thank my family, especially when I remember back to 1989. I cannot think of a family better qualified for supporting someone through a thesis. I dedicate this thesis to them: to my brothers John and Andrew, to my Mum Gillian, and to the memory of my Dad, Peter.





# Contents

<b>1</b>	<b>Introduction</b>	<b>1</b>
1.1	Basic concepts and definitions . . . . .	5
1.1.1	Trees . . . . .	5
1.1.2	Subtrees . . . . .	7
1.1.3	Rooting and unrooting . . . . .	7
1.1.4	Splits and Clusters . . . . .	8
1.1.5	Quartets, rooted triples and fans . . . . .	8
1.2	The scourge of complexity . . . . .	10
<b>2</b>	<b>Compatibility and the Information in Trees</b>	<b>15</b>
2.1	Introduction . . . . .	15
2.2	Introducing Compatibility . . . . .	16
2.2.1	Contraction . . . . .	16
2.2.2	Pruning and Induced Subtrees . . . . .	17
2.2.3	Compatibility Definitions . . . . .	19
2.3	Tree Characterisations and Compatibility . . . . .	21
2.3.1	Quartets and Rooted Triples . . . . .	21
2.3.2	Splits and Clusters . . . . .	22
2.3.3	Nestings . . . . .	22
2.3.4	$n$ -taxon Statements . . . . .	23
2.3.5	Partial Order on Pairs . . . . .	24
2.3.6	Direct Extensions . . . . .	25
2.4	When is a set of unrooted trees compatible? . . . . .	26
2.4.1	Splits and Quartets . . . . .	26
2.4.2	Compatible Splits . . . . .	29

2.4.3	So how can we build trees? . . . . .	33
2.5	When is a set of rooted trees compatible? . . . . .	35
2.5.1	Characterisation of Compatibility . . . . .	37
2.5.2	What is so special about the ONETREE tree? . . . . .	38
2.6	Some rooted tree problems that aren't easy . . . . .	39
2.6.1	Maximum compatible subset of a rooted triple set . . . . .	40
2.6.2	Forbidden Triples . . . . .	41
2.6.3	Maximum Resolved Tree . . . . .	45
<b>3</b>	<b>Building Trees - Inference Rules</b>	<b>49</b>
3.1	Introduction . . . . .	49
3.2	Introduction to inference rules and closed sets . . . . .	50
3.3	Properties of Closed Sets . . . . .	52
3.3.1	Applications of closed sets . . . . .	55
3.4	Closed sets and Rooted Triples . . . . .	59
3.4.1	Graphical characterisation of closure . . . . .	59
3.4.2	Properties of closed sets of rooted triples . . . . .	62
3.4.3	Irreducible Rooted Triple Rules . . . . .	63
3.5	Closed sets and Quartets . . . . .	71
3.5.1	Properties (and non-properties) of Quartet Sets . . . . .	71
3.5.2	Links between Quartets and Rooted Triples . . . . .	72
3.5.3	Irreducible Quartet Rules . . . . .	73
<b>4</b>	<b>Hunting for Trees - Character Data</b>	<b>77</b>
4.1	Introduction . . . . .	77
4.2	A special case: sets of clusters . . . . .	78
4.2.1	Principal algorithm . . . . .	79
4.2.2	Counting the Trees . . . . .	81
4.2.3	Tree extraction . . . . .	83
4.2.4	Optimisation . . . . .	85
4.2.5	Selecting a degree bound . . . . .	87
4.2.6	Hunting through all possible clusters . . . . .	88
4.3	Sets of binary characters . . . . .	89
4.4	Decomposition table bag of tricks . . . . .	91

4.4.1	Intersection of Decomposition tables . . . . .	92
4.4.2	Forcing Characters to be included . . . . .	93
4.4.3	Triples, fans and quartets in common . . . . .	97
4.4.4	Automatic consensus trees . . . . .	102
4.5	Application to Split Decomposition and Quartet Puzzling . . . . .	103
4.5.1	Split Decomposition . . . . .	103
4.5.2	Quartet Puzzling . . . . .	105
4.6	Quartet Compatibility . . . . .	106
4.7	Application to large data sets - human mtDNA . . . . .	108
4.7.1	Framework for investigation . . . . .	108
4.7.2	Analysis of the entire set . . . . .	109
4.7.3	Testing the placement of the !Kung sequences . . . . .	109
4.7.4	Testing the placement of the African sequences . . . . .	110
<b>5</b>	<b>Hunting for Trees - Distance Data</b>	<b>113</b>
5.1	Introduction . . . . .	113
5.2	Trees to Distances and Back . . . . .	114
5.2.1	Three and four point conditions . . . . .	114
5.2.2	Making a space for trees . . . . .	116
5.2.3	Tree building methods . . . . .	118
5.2.4	Optimisation Criterion . . . . .	119
5.3	Estimating Edge Lengths . . . . .	121
5.3.1	Estimation criteria . . . . .	121
5.3.2	The first speed-up: calculating $A^T d$ . . . . .	124
5.3.3	Fast algorithms for OLS, WLS and GLS . . . . .	126
5.4	An unusually fast algorithm for OLS . . . . .	127
5.4.1	Speeding up matrix inversion . . . . .	128
5.4.2	Internal edge length formula . . . . .	131
5.4.3	External edge length formula . . . . .	134
5.4.4	Simplifications for binary trees . . . . .	135
5.4.5	The algorithm . . . . .	137
5.5	ME trees in splits . . . . .	140
5.5.1	Method . . . . .	141

5.5.2	Verification that the method actually works . . . . .	144
5.5.3	Complexity . . . . .	148
5.5.4	Application to local optimisation . . . . .	149
<b>6</b>	<b>Comparing Trees - Consensus and Agreement</b>	<b>155</b>
6.1	Introduction . . . . .	155
6.2	A Survey of Consensus Methods . . . . .	156
6.2.1	Strict Consensus Tree . . . . .	156
6.2.2	Majority Rule Tree . . . . .	157
6.2.3	Loose Consensus Tree . . . . .	157
6.2.4	Adams Consensus . . . . .	157
6.2.5	Aho <i>et al.</i> Consensus Tree . . . . .	159
6.2.6	Nelson Consensus Tree . . . . .	160
6.2.7	Cluster Height Methods . . . . .	160
6.2.8	Median Consensus Tree . . . . .	161
6.2.9	Average Consensus Trees . . . . .	162
6.2.10	A Consensus Classification . . . . .	162
6.3	Consensus trees via axioms . . . . .	164
6.4	Clique Consensus methods . . . . .	167
6.4.1	Complexity . . . . .	168
6.4.2	Restricted Max Clique Consensus . . . . .	169
6.5	Agreement Methods . . . . .	171
6.5.1	Reduced Consensus Profiles . . . . .	171
6.5.2	Agreement subtrees . . . . .	174
6.5.3	Maximum Agreement Subtree (MAST) . . . . .	179
6.5.4	Maximum Information Agreement Subtree (MIST) . . . . .	182
6.5.5	Maximum Number of Triples Agreement Subtree (MANTAT) . . . . .	184
6.5.6	Maximum Number of Quartets Agreement Subtree . . . . .	188
	<b>Bibliography</b>	<b>189</b>
	<b>Appendices</b>	

<b>A</b>	<b>A List of NP-complete problems</b>	<b>201</b>
A.1	General Compatibility Problems . . . . .	201
A.2	Maximum Compatible Subset Problems . . . . .	202
A.3	Forbidden Subtrees Problems . . . . .	203
A.4	Consensus Tree Problems . . . . .	204
A.5	Perfect Phylogeny and Tandyfications . . . . .	205
A.6	Parsimony . . . . .	206
A.7	Distance Based Methods . . . . .	207
A.8	Open Problems . . . . .	207
<b>B</b>	<b>List of Algorithms</b>	<b>209</b>
<b>C</b>	<b>List of Symbols</b>	<b>211</b>



# Chapter 1

## Introduction

Phylogenetics needs mathematicians. It needs them for handling large data sets. It needs them for developing new methods, and analysing old ones. It needs them because mathematics is the art of being systematic.

Advances in molecular biology have led to an explosion in the amount of data available for analysis, and in the size of these data sets. It is no longer possible to assemble data and build trees by hand. There is a growing need for sophisticated new techniques to analyse and understand the data, balanced by the need to recognise which problems are simply too big and too difficult for our current resources.

Mathematicians are not only good for solving overgrown problems. A mathematician or statistician brings a new degree of rigour, or perhaps we should say a new type of rigour, to the field. Mathematicians can turn vague guidelines into systematic methods. They can break a technique or model down into its basic structures and building blocks for the purpose of critique, or extension, or to demonstrate equivalence to some other technique. They can help identify when a result is significant, and when it is simply the outcome of systematic error.

There is a two way exchange, though. Ask not what mathematics can do for biology, but what biology can do for mathematics. Research into phylogenetics has given birth to a new field of mathematics that is only slowly crystallising. One part of it has been given the name ‘T-theory’ by Dress *et al.* [43]. It seems that the theory of quartets and splits that we extend in this thesis is a new type of mathematics, with only loose connections to graph theory, algebra and logic.

However the emphasis of this thesis is on application. In each part of the research

the final product tends to be a useful, practical technique or algorithm, rather than a theorem or observation. It is with this in mind that we conclude the first chapter with a brief discussion of computational complexity. It is not enough to be able to encode an algorithm on a computer: we must know whether the program will be completed by the due date of the research report, or indeed in the lifetime of the universe.

In Chapter 2, **Compatibility and the Information in Trees**, we discuss what kinds of information are contained in a phylogenetic tree, with particular emphasis on the problem of determining when one tree contains all the information contained in another tree or set of trees. The meanings ascribed to facets of a tree diagram have been a major source of conflict between the various schools of systematics. The underlying mathematics in each school, from taxonomy to cladistics to evolutionary systematics, is almost identical. We take a mathematical approach.

The definitions and framework for this research come from the biomathematics community, in particular the work of Bunemann [28], Estabrook *et al.* [46], McMorris [80], Estabrook and McMorris [47], Bandelt and Dress [9], Dekker [40], Dress and Steel [42] and Steel [101]. It also extends work on phylogenetics algorithms for the discussion on compatibility [22, 33, 88, 113]. We introduce new characterisations of compatibility, explore the mathematics of trees, subtrees, quartets and splits, and prove related NP-completeness results. Though there is always a concern for direct biological applications, and assembling many trees into one tree is a real biological problem, the main contribution of this chapter is a theoretical framework for the remaining chapters.

The third chapter, **Building Trees—Inference Rules**, is the most unashamedly theoretical chapter of the thesis. We develop further the mathematics of quartets and trees, greatly extending the work of Dekker [40] on inference rules and the quartet results in [9] and [101]. We introduce and study closed sets, in both the rooted and unrooted contexts, and show that closed sets and inference rules exhibit a surprising level of complexity. The mathematics of quartets is particularly elusive, and we raise several intriguing unanswered questions. This is applied mathematics in the long term sense. It is an exploration of a new area of mathematics, albeit an area with considerable potential for useful application.

The main result of Chapter 4, **Hunting for Trees—Character Data**, began



life as solution to an innocuous theoretical problem. This solution turned out to be directly applicable to character analysis, subsequently leading to a whole host of interesting and useful applications.

Given a set of binary characters we show how to determine whether the set contains enough characters to build trees with bounded degree. We can also find which of these trees has maximum summed edge weight. Our algorithm is applied to the analysis of large data sets, in particular the human mtDNA data set, as well as to Split Decomposition, Quartet Puzzling, Quartet Compatibility, Minimum Evolution trees (in Chapter 5), and consensus problems (in Chapter 6).

The problems solved in Chapter 5, **Hunting for Trees—Distance Data**, come directly from the biology literature, starting with the problem of simplifying the mathematics in [95]. There is a brief survey of distance based tree building methods, and an exploration of links with linear algebra. We describe a very fast algorithm for calculating edge lengths, a hybrid algorithm of Minimum Evolution and Hunting for Trees, and an improved local optimisation method for the minimum evolution problem.

In Chapter 6, **Comparing Trees—Consensus and Agreement**, we address an important biological problem: given a set of conflicting trees on the same leaf set, find a tree that best represents all of them. This is the consensus tree problem and has multiple applications in systematics. We discuss and classify existing methods, uncovering new links between the consensus techniques. We introduce new consensus methods designed for edge weighted trees, along with polynomial time algorithms to construct them.

It may be that the most representative trees do not have the same leaf sets as the input trees. This is the approach taken with agreement subtree methods. Agreement subtrees have been the subject of considerable interest, particularly among computer scientists. Our contribution is a simplified algorithm for the Maximum Agreement subtree problem and polynomial time algorithms for variations of Maximum Agreement Subtree that overcome deficiencies of the Maximum Agreement Subtree method.

To summarise, we list the major results of this thesis.

- Fundamental results in the new mathematics of trees, quartets and splits (Chapters 2 and 3).

- A proof that there are irreducible quartet inference rules of any order (Chapter 3).
- A polynomial time algorithm for determining whether a set of splits contains the splits of a tree with fixed degree bound, as well as polynomial time algorithms for finding which of these trees has maximum summed weight (Chapter 4).
- An optimal  $O(n^2)$  time algorithm for calculating edge weights under ordinary least squares (OLS) for a binary or non-binary tree with  $n$  leaves. An  $O(n^3)$  time algorithm for the same problem with weighted least squares (WLS) and an  $O(n^4)$  time algorithm for generalised least squares (GLS) where the inverse of the weighting matrix is given (Chapter 5).
- A polynomial time algorithm for finding the binary tree with the smallest minimum evolution score over all binary trees with splits contained in a given set of splits (Chapter 5).
- An  $O(n^{2^{4.5r-6}})$  time algorithm for finding a binary tree with a minimum evolution score at least as small as the score for any binary tree  $T'$  within symmetric difference distance  $r$  from a given binary tree  $T$  (Chapter 5).
- A consensus method with polynomial time construction algorithm that takes account of edge weightings in the input set (Chapter 6).
- A simplified algorithm for the Maximum Agreement Subtree problem with bounded degree (Chapter 6).
- A polynomial time algorithm that finds a bounded degree agreement subtree with the largest number of internal edges (Chapter 6).
- A polynomial time algorithm that finds a bounded degree agreement subtree compatible with the largest number of rooted triples (Chapter 6).
- NP-completeness results for a wide range of phylogenetics problems (Chapters 2–6 and Appendix A).

## 1.1 Basic concepts and definitions

### 1.1.1 Trees

This thesis is all about trees, in particular the trees used in phylogenetic analysis. It is therefore appropriate to start everything off with the formal definition of a tree.

We define an **unrooted phylogenetic tree** or **unrooted leaf-labelled tree** or just **unrooted tree** to be an acyclic connected graph with no vertices of degree two and every **leaf** (vertex of degree one) labelled uniquely. Internal vertices (vertices that are not leaves) are usually left unlabelled. This corresponds to a phylogenetic tree in [42, 101], to a semilabelled tree in [103], an S-labelled tree in [7] and a fully resolved tree structure in [9].

A **rooted phylogenetic tree** is defined in the same way, except that one internal vertex, which can have degree two, is distinguished and called the **root**. The remaining internal vertices all have degree three or greater. In this thesis, the root will be labelled  $\rho$ .

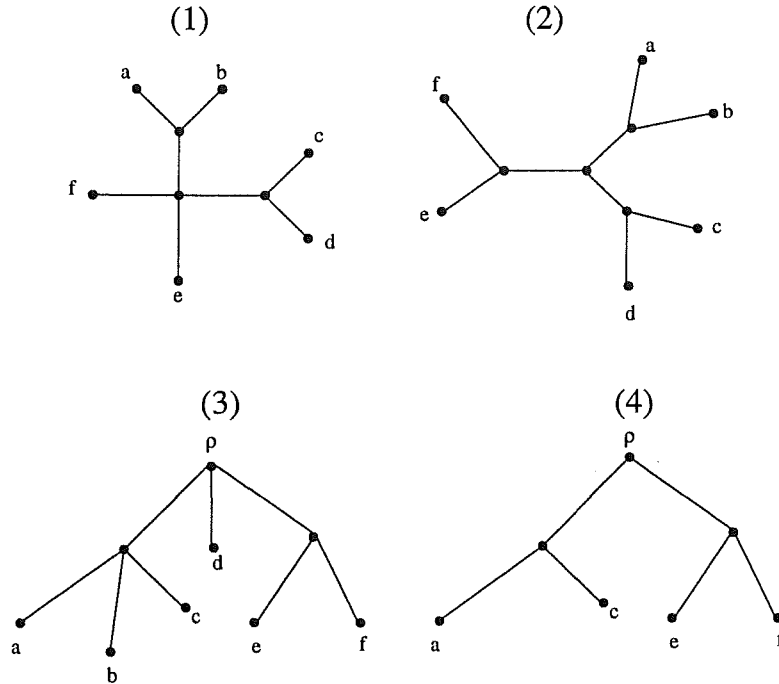


Figure 1 : Four examples of *phylogenetic trees*. (1) and (2) are *unrooted*. (3) and (4) are *rooted*. (2) and (4) are *binary*.

In a **binary unrooted phylogenetic tree** every internal (i.e. non-leaf) vertex has degree three. This is called a non-degenerate tree structure in [9]. In a **binary rooted phylogenetic tree**, all internal vertices have degree three, except the root which has degree two. In an **unrooted  $d$ -tree** every vertex has degree at most  $d$ , where the degree of a vertex equals the number of incident edges. In a **rooted  $d$ -tree** every vertex has at most  $d$  children.

One standard example of a binary tree is the **caterpillar tree**. An unrooted caterpillar tree has one central path with leaves branching off it, like tree (1) of Figure 2. The rooted caterpillar has leaves appended to a single path from the root to a single leaf, like tree (2) of Figure 2.

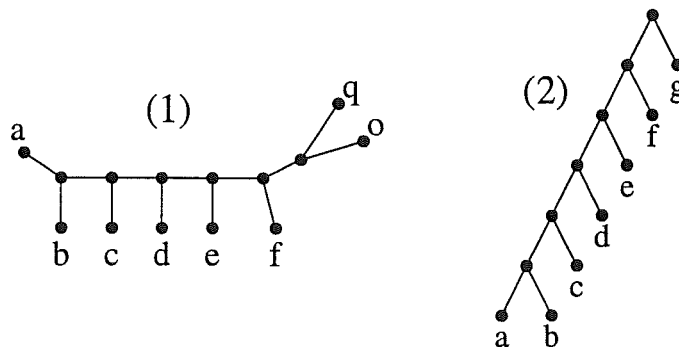


Figure 2 : Unrooted and rooted caterpillar trees.

Given any tree  $T$ , let  $\mathcal{L}(T)$  be the leaf set of  $T$ . If  $\mathcal{T}$  is a set of trees, let  $\mathcal{L}(\mathcal{T})$  be the union of the leaf sets of the trees in  $\mathcal{T}$ .

If two leaves in a rooted or unrooted tree are adjacent to a vertex that is not adjacent to any other other leaves then the pair of leaves is called a **pendant pair**. A rooted binary tree with more than one leaf has at least one pendant pair, and an unrooted binary tree with four or more leaves has at least two. Caterpillar trees have the minimum number of pendant pairs.

A vertex  $a$  in a rooted tree is a **descendant** of a vertex  $b$  if the path from  $a$  to the root  $\rho$  passes through  $b$ . In this case, we say that  $b$  is an **ancestor** of  $a$ . The vertices adjacent to a vertex that are descendants of the vertex are called the **children** of the vertex, and an adjacent vertex that is an ancestor is called the **parent** of that

vertex. The **lowest common ancestor** of a set of vertices  $X$  is the unique ancestor of  $X$  that is a descendant of all the ancestors of  $X$ .

Sometimes the internal vertices of a phylogenetic tree are labelled, or a vertex might have more than one label [40, 42]. (These trees are also called ‘S-labelled Trees’ [113], or ‘Tree Structures’ [9]).

Rooted phylogenetic trees can be displayed with a vertical axis representing the time each branching point occurred. These diagrams are called dendrograms. In this thesis we are only concerned with the underlying branching tree structure.

### 1.1.2 Subtrees

Let  $e_1$  be an internal edge in an unrooted tree  $T$ , and let  $\alpha$  and  $\beta$  be the endpoints of  $e_1$ . Choose any other edge adjacent to  $\alpha$  and remove it, giving two connected subgraphs. If we root the subgraph not containing  $e_1$  at the point of the cut then we obtain the **subtree of  $T$  branching off  $e_1$  at  $\alpha$** . We sometimes examine the set of such subtrees.

Now let  $T$  be a rooted tree and choose a vertex  $v$  in  $T$ . Removing the edge between  $v$  and the parent of  $v$  gives two connected subgraphs. Root the subgraph containing  $v$  at the vertex  $v$ , this is then the **subtree of  $T$  rooted at  $v$** . Let  $w_1, \dots, w_k$  be the children of  $v$ . The subtrees of  $T$  rooted at  $w_1, w_2, \dots, w_k$  are called the **subtrees of  $T$  branching off at  $v$** . If  $\rho$  is the root of  $T$  then the subtrees of  $T$  branching off at  $\rho$  are called the **maximal subtrees of  $T$** .

### 1.1.3 Rooting and unrooting

There is a natural correspondence between rooted trees and unrooted trees. Given a rooted tree  $T$  and a new leaf  $x$ , append  $x$  to the root of  $T$  and ‘unroot’ the tree, that is, construct the unrooted tree with the same underlying graph. We denote this tree by  $\text{ROOT}(T, x)$ . The operation is invertible. Given an unrooted tree  $T'$  with a leaf  $x$ , root  $T'$  at the vertex adjacent to  $x$  and then remove  $x$  together with its incident edge. This tree is denoted  $\text{UNROOT}(T', x)$  (Figure 3).

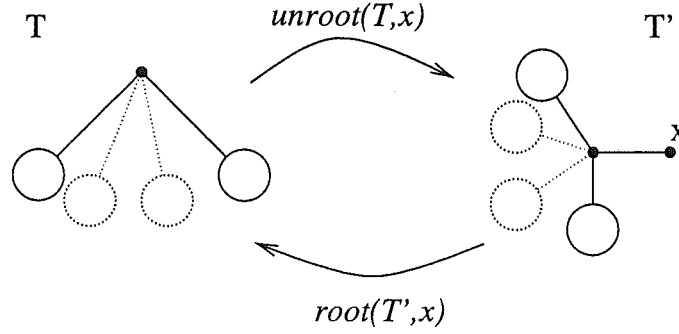


Figure 3 : The operations  $ROOT()$  and  $UNROOT()$ . The circles denote subtrees.

#### 1.1.4 Splits and Clusters

Let  $T$  be an unrooted tree and let  $e$  be an edge of  $T$ . If we remove  $e$  then we divide up  $T$  into two components. Let  $A$  be the leaves in one component and  $B$  be the leaves in the other component. Then  $A|B$  is a partition of  $\mathcal{L}(T)$  into two blocks, called a **split** or **bipartition** of  $\mathcal{L}(T)$ . The split  $A|B$  is said to be the **split corresponding to the edge  $e$** . The set of those splits corresponding to edges in  $T$  is called the **set of splits of  $T$**  or just the **splits in  $T$**  and is denoted  $\beta(T)$ . We say that a **split  $A|B$  is in  $T$**  if  $A|B$  corresponds to an edge of  $T$ . If  $|A| = 1$  or  $|B| = 1$  then  $A|B$  is **trivial**, otherwise it is **non-trivial**. The trivial splits correspond to external edges. A tree can be reconstructed in linear time from its set of splits [29, 83, 60].

Now consider a rooted tree  $T$ . If we choose a vertex  $v$  then the set of leaves in  $T$  that are descendants of  $v$  is called a **cluster**. The set of those clusters corresponding to vertices of  $T$  is called the **set of clusters of  $T$**  and is denoted  $\sigma(T)$ . We say that  $A$  is a **cluster in  $T$**  if  $A \in \sigma(T)$ . A cluster  $A$  is **trivial** if  $|A| = 1$  or  $A = \mathcal{L}(T)$ . The **maximal clusters** of a tree  $T$  are the maximal *non-trivial* clusters in  $\sigma(T)$ , that is, the leaf sets of the maximal subtrees. Note that a rooted tree is uniquely defined by its clusters, and can be reconstructed from these in linear time [60].

#### 1.1.5 Quartets, rooted triples and fans

For every three leaves  $a, b, c$  there is only one unrooted tree with leaf set  $\{a, b, c\}$ , though there are four unrooted trees for any set of four leaves (Figure 4). The three

binary trees with four leaves are called **quartets**. The quartet with two pendant pairs  $\{a, b\}$  and  $\{c, d\}$  is denoted  $ab|cd$ .

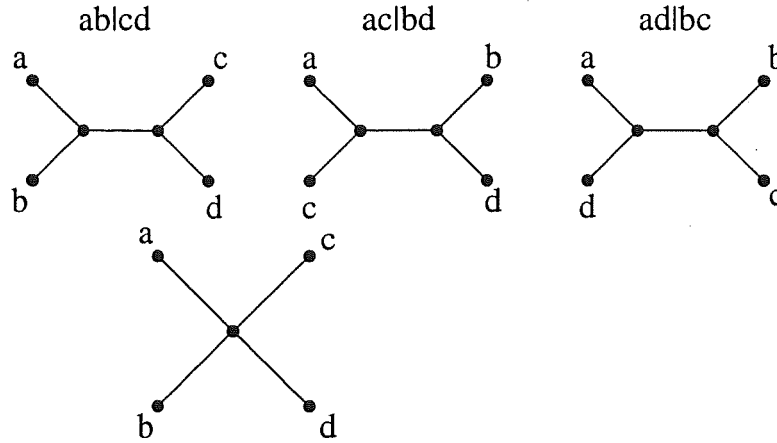


Figure 4 : The four unrooted trees with four leaves.

We say that a quartet  $ab|cd$  fits a tree  $T$  if the path from  $a$  to  $b$  in  $T$  does not share any vertices with the path from  $c$  to  $d$  in  $T$ . The **quartet set of a tree** or just the set of **quartets in a tree** is the set of quartets that fit  $T$ , and is denoted  $q(T)$ . Any tree with at least one internal edge can be reconstructed from its quartet set [9].

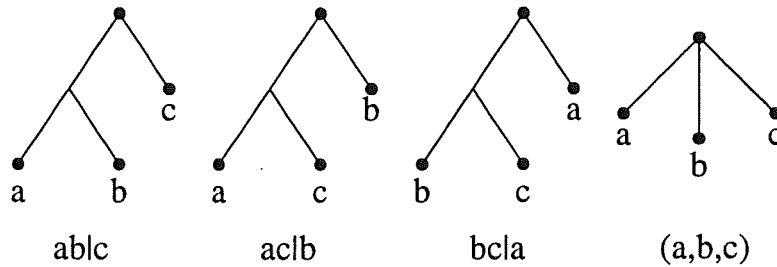


Figure 5 : The four rooted trees with three leaves.

For every three leaves  $a, b, c$  there are four rooted trees with leaf set  $\{a, b, c\}$  (Figure 5). The binary rooted trees on three leaves are called **rooted triples** and

$ab|c$  denotes the rooted triple with a pair of leaves  $\{a, b\}$  connected to a third leaf  $c$  via the root. A rooted triple **fits** a rooted tree  $T$  if the path from  $a$  to  $b$  does not share any vertices with the path from  $c$  to the root. The set of rooted triples that fit are tree is denoted  $r(T)$  and called the **rooted triple set** of  $T$ , or just the set of **rooted triples in  $T$** .

Non-binary rooted trees with three leaves are called **fan triples**. We let  $(a, b, c)$  denote the fan triple with leaf set  $\{a, b, c\}$ . A fan triple  $(a, b, c)$  **fits** a tree if the lowest common ancestors of  $a, b$  and  $a, c$  and  $b, c$  are equal. The set  $f(T)$  is the set of fan triples that fit a rooted tree  $T$ .

## 1.2 The scourge of complexity

One of the declared aims of this thesis is the construction of practical algorithms for solving problems in phylogenetics. The algorithms must not only work correctly, they must be fast enough to be useful. There is little point implementing an algorithm that will take millions of years to run.

Computational complexity is a measure of the efficiency of an algorithm or the difficulty of a problem, principally in terms of the time and resources required. Throughout this thesis we will be discussing the complexity of problems and algorithms in phylogenetics, so it is appropriate that we review the basic principles. Refer to [55, 37, 112, 59] for more detailed surveys of complexity theory.

A function  $f$  is  $O(g(n))$  if there exists  $r > 0$  such that  $f(n) < r \cdot g(n)$  for all but finitely many  $n$ . We say that an algorithm takes  $O(g(\cdot))$  **time** if the maximum possible number of operations it requires is  $O(g(\cdot))$ , where  $g$  is a function of the parameters of the algorithm. The **complexity of a problem** is defined to be the complexity the most efficient algorithm that solves it.

A **polynomial time** algorithm has complexity  $O(g(\cdot))$  for some finite degree polynomial  $g$ . Polynomial time algorithms generally run faster than non-polynomial time algorithms. Compare, for example, the rate of growth of  $n^2$  to the rate of growth of  $2^n$ . For this reason problems with polynomial time solutions are said to be **tractable** while those without polynomial time solutions are **intractable**. The set of polynomial time decision problems is denoted  $P$ . A decision problem is one that has an answer of “yes” or “no”.



There are two parts to solving any problem: finding a solution and checking that it is valid. The set of problems with polynomial time algorithms for the second part is denoted NP, short for non-deterministic polynomial. For example, the perfect phylogeny problem is in NP because we can quickly test whether a given tree is a perfect phylogeny for the set of input characters. Clearly  $P \subset NP$ . It is not currently known whether  $P \neq NP$ , but it is generally accepted.

Another important concept in complexity theory is that of **polynomial reducibility**. A **polynomial transformation or reduction** from one problem into another problem is a polynomial time algorithm for solving the first problem that assumes there is a polynomial time algorithm for the second problem. So if there is a polynomial time transformation from problem  $A$  into problem  $B$  then a polynomial time algorithm for  $B$  will give a polynomial time algorithm for  $A$ . Hence finding a polynomial time algorithm for  $B$  is at least as hard as finding a polynomial time algorithm for  $A$ .

In 1971, Cook [34] showed any problem in NP could be transformed to a problem called the “satisfiability problem”. If the “satisfiability” problem can be solved in polynomial time then so can all problems in NP. Hence if any problem in NP does not have a polynomial time solution then “satisfiability” problem does not have a polynomial time solution. There are hundreds of well-known problems that share this characteristic, including problems in graph theory, logic, programming, language theory and, as we will show, phylogenetics. These problems are called **NP-complete**. They are the hardest problems in NP.

Though it has not been proven, it is generally accepted that NP-complete problems do not have polynomial time solutions. The justification for this assumption is that, so far, thousands of talented researchers have failed to find polynomial algorithms for these problems. Moreover a polynomial time algorithm for one NP-complete problem implies a polynomial time algorithm for all NP-complete problems. As more and more problems are added to the list of NP-complete algorithms this eventuality becomes less and less likely.

The theory of NP-completeness is a useful tool. If we can show that a problem is NP-complete then we need not waste time searching for a polynomial time solution, under the assumption that  $NP \neq P$ .

There are two steps to proving that a problem is NP-complete. First we have to

show that the problem is in NP. This is typically not difficult. Second we have to describe a transformation of a known NP-complete problem into the given problem. That is, we have to prove that if our problem has a polynomial time algorithm then an NP-complete problem has a polynomial time algorithm, and hence all NP-complete problems have polynomial time algorithms. This step is usually tricky.

A related class of problems is the set of NP-hard problems. A problem is **NP-hard** if a polynomial time algorithm for the problem gives a polynomial time algorithm for some NP-complete problem. The difference is that NP-hard problems do not have to be in NP. This includes many problems that are not decision problems, but cannot be solved in polynomial time unless  $P=NP$ .

Special consideration must be given to problems involving real numbers. Many irrational numbers are not computable, meaning that they cannot be represented in a finite string so must be approximated for computer calculation. Hence problems with real numbers are not formally members of the classes NP and P. For reasons of simplicity, we choose to ignore this fact during this thesis. Technically we assume that all real numbers are approximated infinitely closely by a computable rational and that all rational numbers take only one unit of memory.

Phylogenetics is full of NP-complete problems, many of which we list in Appendix A. This tendency towards intractability is depressing. Polynomial time algorithms for problems are much more useful than proofs on NP-completeness. However, as noted by [55], proving a problem NP-complete does not make the problem go away. It just calls for a change of strategy. Three possible approaches are:

(i) Modify the problem. This is not an unrealistic option. Often we can place bounds on some of the parameters. The Perfect Phylogeny Problem is NP-complete, but can be solved in polynomial time with a bound on the number of character states [69, 4, 67]. This bound is well suited to the analysis of four state genetic data. In Chapter 4 we modify the NP-hard Maximum Compatible Subset Algorithm by putting a degree bound on the output tree.

Note that even though a problem might be polynomial when one parameter is fixed we might still not get a practical solution. There are different degrees of difficulty even among problems that become polynomial with a bounded parameter. A problem that has complexity  $O(n^m)$ , where  $n$  is large, is generally much more difficult to solve than a problem with complexity  $O(n2^m)$ . The theory of parameterised

complexity explores which problems have these types of solutions [41].

(ii) Heuristic or approximation algorithm. We may not be able to solve a problem exactly, but we can at least get very close. Heuristic algorithms find good solutions, rather than the ‘best’ solutions. They find, for example, a local optimum instead of a global optimum. This is often the most practical approach to an intractable problem. However there are some problems, like the Maximum Agreement Subtree problem, that are so hard that even approximating a solution is NP-hard [63].

(iii) The third approach is to go ahead and use whatever algorithm you have, even if it is not polynomial. This is the default approach of phylogenetics practitioners. Naturally this restricts the size of the problems that can be tackled, but sometimes that is not too much of a constraint. While two weeks of computing time might be intolerable for a algorithm designer obsessed with efficiency, it is totally reasonable to a researcher that has spent several years, and large research grants, collecting the data.



## Chapter 2

# Compatibility and the Information in Trees

### 2.1 Introduction

Evolutionary trees contain information and some trees are more informative than others.

These are trivial observations, but together they motivate the study of compatibility and, particularly, the study of when one tree contains all the information present in another tree. They also force us to address the question: “What is the information contained in a tree?”

If we take an ‘applied’ perspective then the answer to this question depends on one’s school of phylogenetics. To a cladist, the information in a tree (cladogram) is the set of nested groups or clades. To a evolutionary biologist the information in a tree is the evolutionary history it describes, particularly the various common ancestors of different species. In other contexts a tree is simply a convenient representation of proximity or degree of similarity.

Taking a ‘theoretical’ perspective we would also find a number of different ways to describe the information in a tree. We have already seen (Chapter 1, section 1.1) that an unrooted tree can be characterised as a partially labelled acyclic connected graph, or a collection of splits, or a set of quartets. Likewise, a rooted tree can be described in terms of a graph, a collection of clusters, or a set of rooted triples. In addition we will also present a nested set characterisation and a partial order on

pairs of leaves characterisation.

Before that we define the concepts of extension and compatibility. The original definitions are made in terms graph theoretic operations. We show how these concepts translate over to the various tree characterisations.

We discuss the problem of determining whether there exists a tree that extends each tree in given set of trees. In the *unrooted* case, this problem has already been shown to be NP-complete [101, 22]. We extend this result by showing that even determining whether there is a split compatible with a set of quartets is NP-complete. Accepting that a polynomial time algorithm for the problem is unlikely, we investigate exponential time algorithms.

In the *rooted* case, the problem of determining whether there exists a rooted tree that extends a set of trees can be solved in polynomial time using an algorithm dating back to 1981 [6]. The algorithm forms the basis of a new graphical representation of rooted triple sets. We can describe compatibility in terms of this graphical representation (Theorem 2.14).

Even though rooted tree compatibility is easy there are several variations of the problem that are not so easy. We show that finding a maximum compatible subset of rooted triples is NP-Hard, as is finding a tree not compatible with any of a given set of rooted triples, and also finding the most resolved tree that extends a set of rooted triples and fans.

## 2.2 Introducing Compatibility

Extension and compatibility are based on two operations on trees, edge contraction and pruning.

### 2.2.1 Contraction

Let  $T$  be a rooted or unrooted tree. A **contraction** of  $T$  is obtained by deleting an internal edge and identifying its endpoints. Repeating the process for other edges gives additional contractions.

Consider Figure 6. The horizontal edge in the tree on the left is contracted to give the tree in the centre. The process is repeated to give another contraction on

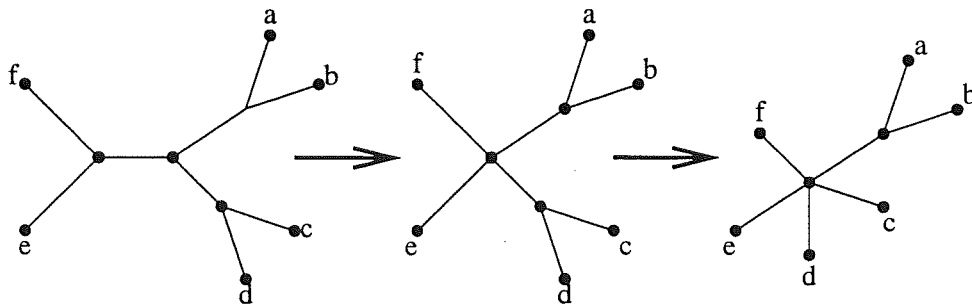


Figure 6 : An example of edge contraction.

the right. If all the internal edges are contracted then the resulting tree will be the star tree with one internal vertex and edges from the central vertex to all the leaves.

The converse of contraction is refinement. A tree  $T$  **refines** a tree  $S$  if and only if  $S$  is a contraction of  $T$  [42, 101].

Contraction gives a natural partial order on the set of unrooted trees on a given leaf set. A tree  $S$  is a descendant of a tree  $T$  if and only if  $S$  is obtained by contracting edges of  $T$ . It was observed in [15] that this partially ordered set is a meet-semi-lattice and the meet of two trees is equal to their strict consensus tree (see Chapter 6).

If we restrict the set to those trees that are contractions of some tree  $T$  then the poset becomes a boolean algebra, since there is a one to one correspondence between contractions of a tree and subsets of its internal edge set.

### 2.2.2 Pruning and Induced Subtrees

Let  $T$  be a rooted or unrooted tree, and let  $A$  be a subset of its leaf set  $\mathcal{L}(T)$ . Consider the minimal subgraph  $T(A)$  of  $T$  that connects elements of  $A$ . If  $T$  is unrooted then delete all vertices of degree two in  $T(A)$  and identify their adjacent edges, thereby obtaining an unrooted tree with leaf set  $A$ , denoted  $T|_A$ . If  $T$  is rooted then we distinguish the vertex of  $T(A)$  that is closest to the root in  $T$  and make it the new root, and then delete any remaining vertices in  $T(A)$  of degree two,

identifying their adjacent edges. We obtain a rooted tree with leaf set  $A$ , denoted  $T|_A$ . In both cases the tree  $T|_A$  is called the **subtree of  $T$  induced by  $A$** . The process of removing leaves not in  $A$  is called **pruning** [107, 53].

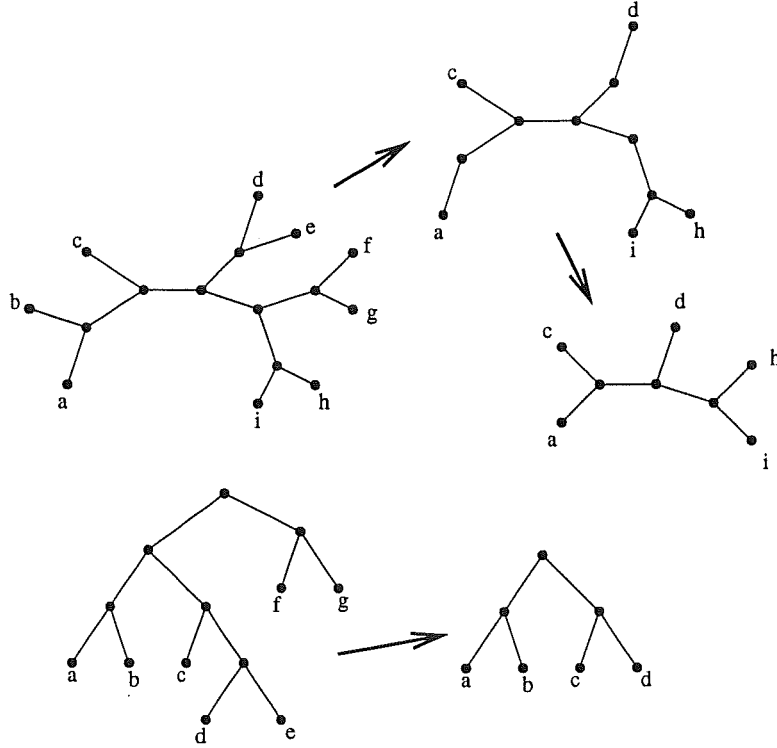


Figure 7 : Induced subtrees of unrooted and rooted trees. In the unrooted case we include the intermediary step. Note that in the rooted case the root of the induced subtree is different from the root of the original tree.

For example, consider Figure 7. The three stages of obtaining an induced subtree are outlined. Note that in the rooted case the induced subtree had a different root from the original tree.

Other names for the induced subtree  $T|_A$  are the restriction of  $T$  on  $A$  [63] and the homeomorphic subtree of  $T$  on  $A$  [88].

The set of unrooted trees can be partially ordered by making one tree  $S$  a descendant of another tree  $T$  if and only if  $S$  is an induced subtree of  $T$ . The partially ordered set is not a meet-semilattice: the two trees in Figure 8 have no



greatest lower bound. However the set of induced subtrees of a given tree  $T$  do form a boolean algebra. Each induced subtree corresponds to a subset of  $\mathcal{L}(T)$ , the leaf set of  $T$ .

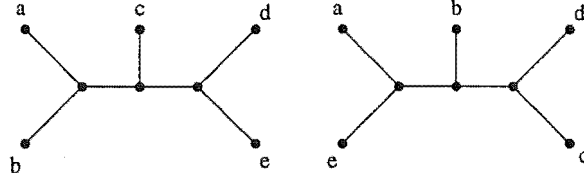


Figure 8 : These two trees have no greatest lower bound under the induced subtree partial ordering and also the compatibility partial order  $\trianglelefteq$ . The quartet  $ab|cd$  and the tree with leaves  $\{a, c, e\}$  are both maximal lower bounds for these trees.

### 2.2.3 Compatibility Definitions

The concepts of extension and compatibility combine contraction and pruning. A tree  $T$  **extends** a tree  $S$ , denoted  $S \trianglelefteq T$ , if  $S$  can be obtained by contractions of an induced subtree of  $T$ , or equivalently, if  $S$  is an induced subtree of a contraction of  $T$ . Clearly  $\trianglelefteq$  gives a partial order on the set of trees. We call it the **compatibility partial order**.

A set  $\mathcal{T}$  of trees is **compatible** if there exists a tree  $T$  that **extends**  $\mathcal{T}$ , that is, if there exists a tree  $T$  that extends every tree in  $\mathcal{T}$ . We say that a tree  $T_1$  is compatible with a tree  $T_2$  if there is a tree that extends them both.

Let  $\mathcal{T}$  be a set of trees and let  $L = \mathcal{L}(\mathcal{T})$  be the union of the leaf sets of trees in  $\mathcal{T}$ . If  $\mathcal{T}$  is a set of rooted trees, then the **span** of  $\mathcal{T}$ , or  $\langle \mathcal{T} \rangle$  is the set of rooted trees with leaf set  $L$  that extend  $\mathcal{T}$ . Likewise, if  $\mathcal{T}$  is a set of unrooted trees, then the span of  $\mathcal{T}$  is the set of unrooted trees on leaf set  $L$  that extend  $\mathcal{T}$ . Hence  $\mathcal{T}$  is compatible if and only if  $\langle \mathcal{T} \rangle \neq \emptyset$ .

The underlying assumption made when choosing this type of compatibility is that the tree structures we are trying to model are binary. A non-binary tree corresponds to incomplete knowledge. If a vertex in an unrooted tree has degree greater than three it is because we are unsure how that vertex is resolved. The same applies for

vertices in rooted trees with more than two descendants. Fan trees are considered to be non-informative.

Compatibility and extension appear under a number of different guises in the literature. Our definition of compatibility is the same as [101], except that we will use the word ‘extends’ where Steel uses ‘compatible’. See [33, 7, 88] for other definitions, some of which ignore contractions and others that do not incorporate pruning.

The set of trees partially ordered by  $\trianglelefteq$  does not form a meet-semilattice: the two trees in Figure 8 have no greatest lower bound under  $\trianglelefteq$ . Given some tree  $T$  the set of trees  $\{S : S \trianglelefteq T\}$  does not form a lattice under  $\trianglelefteq$ . Consider the tree  $T_1$  in Figure 9 and the set of trees  $\{S : S \trianglelefteq T_1\}$  partially ordered by  $\trianglelefteq$ . The two trees  $T_2$  and  $T_3$  have no meet in  $\{S : S \trianglelefteq T_1\}$ : the star tree with leaf set  $\{a, b, c, d, e\}$  and the quartet  $ab|cd$  are both maximal elements of  $\{S : S \trianglelefteq T_2 \text{ and } S \trianglelefteq T_3\}$ .

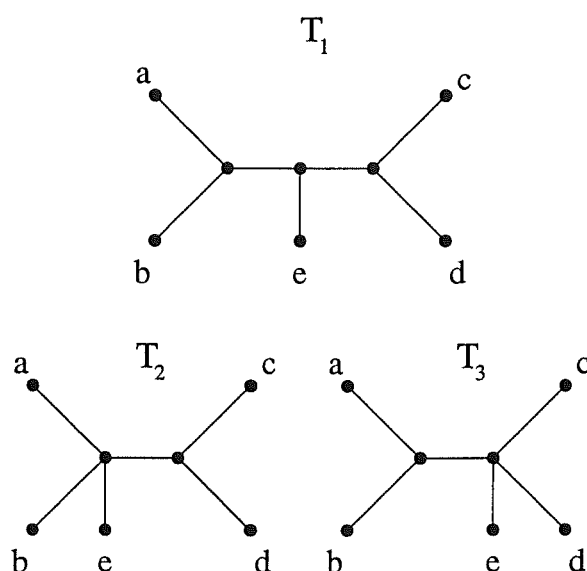


Figure 9 : The two trees  $T_2$  and  $T_3$  do not have a unique meet in the set of trees  $\{S : S \trianglelefteq T_1\}$  partially ordered by  $\trianglelefteq$ .

## 2.3 Tree Characterisations and Compatibility

### 2.3.1 Quartets and Rooted Triples

Earlier (section 1.1.5, page 8) we defined the quartet set of an unrooted tree  $T$  to be the set of quartets  $ab|cd$  such that the path from  $a$  to  $b$  does not intersect the path from  $c$  to  $d$ . Clearly then, a quartet is in the quartet set of a tree  $T$  if and only if  $T$  extends the quartet, and hence  $q(T) = \{ab|cd : ab|cd \trianglelefteq T\}$ . Likewise, if  $T$  is rooted then  $r(T) = \{ab|c : ab|c \trianglelefteq T\}$ .

If unrooted tree  $T$  has any internal edges then  $T$  can be reconstructed from  $q(T)$  [32, 9] so, in some sense,  $q(T)$  contains the ‘information’ of  $T$ . We show that compatibility can be defined in terms of this quartet information.

**Theorem 2.1** *Let  $S$  and  $T$  be unrooted phylogenetic trees.  $T$  extends  $S$ , that is  $S \trianglelefteq T$ , if and only if  $q(S) \subseteq q(T)$  and  $\mathcal{L}(S) \subseteq \mathcal{L}(T)$ . Similarly, let  $S$  and  $T$  be rooted phylogenetic trees.  $S \trianglelefteq T$  if and only if  $r(S) \subseteq r(T)$  and  $\mathcal{L}(S) \subseteq \mathcal{L}(T)$ .*

*Proof*

Suppose first that  $S \trianglelefteq T$ . If  $ab|cd \in q(S)$  then  $ab|cd \trianglelefteq S$ . Since  $\trianglelefteq$  is transitive, we have that  $ab|cd \trianglelefteq T$  and so  $ab|cd \in q(T)$ . The definition of  $\trianglelefteq$  gives  $\mathcal{L}(S) \subseteq \mathcal{L}(T)$ .

Conversely, suppose that  $q(S) \subseteq q(T)$  and  $\mathcal{L}(S) \subseteq \mathcal{L}(T)$ . To prove that  $S \trianglelefteq T$  we need to show that  $S$  is a contraction of  $T|_{\mathcal{L}(S)}$ . Let  $A|B$  be a split of  $S$ . Then  $ab|cd \in q(S)$  for all  $a, b \in A$  and  $c, d \in B$  [9]. Since  $q(S) \subseteq q(T)$  we have  $q(S) \subseteq q(T|_{\mathcal{L}(S)})$  and  $ab|cd \in q(T|_{\mathcal{L}(S)})$  for all  $a, b \in A$  and  $c, d \in B$ . Hence  $A|B$  is a split of  $T|_{\mathcal{L}(S)}$ . By Theorem 1,(1) in [42],  $S$  is a contraction of  $T|_{\mathcal{L}(S)}$ .

An analogous argument applies for the rooted case.  $\square$

Note that  $q(S) \subseteq q(T)$  implies  $\mathcal{L}(S) \subseteq \mathcal{L}(T)$  whenever  $S$  has any internal edges.

A consequence of Theorem 2.1 is that when calculating the span  $\langle \mathcal{T} \rangle$  of a set of trees  $\mathcal{T}$  we can assume without loss of generality that all of the trees in  $\mathcal{T}$  have exactly four leaves. In practice, we can replace each tree  $T \in \mathcal{T}$  by  $q(T)$ , or by a set of quartets that determines  $T$  [101].

### 2.3.2 Splits and Clusters

Given two unrooted trees  $S$  and  $T$  on the same set of leaves,  $S$  is a contraction of  $T$  if and only if  $\beta(S) \subseteq \beta(T)$  [42]. If  $S$  and  $T$  are rooted trees on the same set of leaves then  $S$  is a contraction of  $T$  if and only if  $\sigma(S) \subseteq \sigma(T)$  [26]. We extend these results to characterise compatibility in terms of splits.

For a set of splits  $\mathcal{S}$  on a leaf set  $L$ , and  $X \subseteq L$ , define

$$\mathcal{S}_{|X} := \{A \cap X | B \cap X : A | B \in \mathcal{S}, A \cap X \neq \emptyset, B \cap X \neq \emptyset\}.$$

Note that  $\beta(T_{|X}) = \beta(T)_{|X}$ .

Given a set of clusters  $\mathcal{C}$  of  $L$ , put  $\mathcal{C}_{|X} := \{A \cap X : A \in \mathcal{C}, A \cap X \neq \emptyset\}$ . As with the unrooted case,  $\sigma(T_{|X}) = \sigma(T)_{|X}$ .

**Corollary 2.2** *Let  $S$  and  $T$  be unrooted phylogenetic trees.  $S \trianglelefteq T$  if and only if  $\beta(S) \subseteq \beta(T)_{|\mathcal{L}(S)}$  and  $\mathcal{L}(S) \subseteq \mathcal{L}(T)$ . Similarly, let  $S$  and  $T$  be rooted phylogenetic trees.  $S \trianglelefteq T$  if and only if  $\sigma(S) \subseteq \sigma(T)_{|\mathcal{L}(S)}$  and  $\mathcal{L}(S) \subseteq \mathcal{L}(T)$ .*

*Proof*

$S \trianglelefteq T$  if and only if  $\mathcal{L}(S) \subseteq \mathcal{L}(T)$  and  $S$  is a contraction of  $T_{|\mathcal{L}(S)}$ , and  $S$  is a contraction of  $T_{|\mathcal{L}(S)}$  if and only if  $\beta(S) \subseteq \beta(T_{|\mathcal{L}(S)}) = \beta(T)_{|\mathcal{L}(S)}$ .

An analogous argument applies for the rooted case.  $\square$

### 2.3.3 Nestings

Adams [2] defines a relation  $<_T$  on sets of leaves in a rooted tree  $T$  (see also [116]). Let  $A$  and  $B$  be subsets of  $\mathcal{L}(T)$ . We write  $A <_T B$  if  $A \subset B$  and the lowest common ancestor of  $A$  is a proper descendant of the least common ancestor of  $B$ . In this case we say that  $A$  **nest**s in  $B$ , and  $A <_T B$  is a **nesting** of  $T$ . This nesting order defines the tree uniquely, and Adams [2] presents a characterisation of when a given ordering equals the nesting order of a tree. We show that compatibility corresponds to one tree containing all the nesting information of another tree.

**Corollary 2.3** *Let  $S$  and  $T$  be two rooted phylogenetic trees.  $S \trianglelefteq T$  if and only if  $A <_S B$  implies  $A <_T B$ .*

*Proof*

Note that  $\{a, b\} <_T \{a, b, c\}$  if and only if  $ab|c \in r(T)$ . Suppose that  $A <_S B$  implies  $A <_T B$ . Clearly  $\mathcal{L}(S) \subseteq \mathcal{L}(T)$ . If  $ab|c \in r(S)$  then  $\{a, b\} <_S \{a, b, c\}$  so  $\{a, b\} <_T \{a, b, c\}$  and  $ab|c \in r(T)$ . By Theorem 2.1,  $S \leq T$ .

Conversely suppose that  $S \leq T$ . If  $A <_S B$  then  $A \subset B$  and the least common ancestor of  $A$  is a descendant of least common ancestor of  $B$ . This will still be true if we add leaves and expand contracted vertices. Hence  $A <_T B$ .  $\square$

### 2.3.4 $n$ -taxon Statements

Wilkinson [116] introduces a new class of tree information, the  $n$ -taxon statement. Clusters and rooted triples are both extreme examples of  $n$ -taxon statements. If  $A$  and  $B$  are subsets of  $\mathcal{L}(T)$  for some rooted tree then  $(A)B$  is an  $n$ -taxon statement for  $T$  if  $n = |A \cup B|$  and the leaves in  $A$  are more closely related to each other than any are to leaves in  $B$ . That is, there is some cluster  $X$  in  $\sigma(T)$  such that  $A \subseteq X$  and  $B \cap X = \emptyset$ . A triple  $ab|c$  corresponds to the 3-taxon statement  $(a, b)c$ , and a cluster  $A \in \sigma(T)$  corresponds to the  $n$ -taxon statement  $(A)B$ , where  $B = \mathcal{L}(T) - A$ .

As noted in [116],  $(A)B$  is an  $n$ -taxon statement for  $T$  if and only if  $ab|c \in r(T)$ ,  $\forall a, b \in A$  and  $c \in B$ . We utilise this relationship to characterise compatibility using  $n$ -taxon statements.

**Corollary 2.4** *Let  $S$  and  $T$  be rooted trees.  $S \leq T$  if and only if  $(A)B$  is an  $n$ -taxon statement of  $T$  whenever  $(A)B$  is an  $n$ -taxon statement for  $S$ .*

*Proof*

Suppose that  $(A)B$  an  $n$ -taxon statement for  $T$  for all  $n$ -taxon statements  $(A)B$  of  $S$ . Given  $ab|c \in r(S)$ ,  $(a, b)c$  is an  $n$ -taxon statement of  $S$ , so it is also one of  $T$  and  $ab|c \in r(T)$ . Clearly  $(\mathcal{L}(S))\emptyset$  is an  $n$ -taxon statement of  $S$ , so it is an  $n$ -taxon statement of  $T$  and  $\mathcal{L}(S) \subseteq \mathcal{L}(T)$ . By Theorem 2.1,  $S \leq T$ .

Conversely if  $S \leq T$  and  $(A)B$  is an  $n$ -taxon statement of  $S$ , then  $ab|c \in r(S)$ ,  $\forall a, b \in A$  and  $c \in B$ . Hence  $ab|c \in r(T)$ ,  $\forall a, b \in A$  and  $c \in B$ , so  $(A)B$  is an  $n$ -taxon statement of  $T$ .  $\square$

### 2.3.5 Partial Order on Pairs

Every internal vertex in a rooted tree equals the least common ancestor of at least one pair of leaves. Hence the partial ordering on vertices in a rooted tree given by the descendance relation induces a partial order on pairs of leaves. We write  $\{a, b\} \prec \{c, d\}$  if the least common ancestor of  $a$  and  $b$  is a proper descendant of the least common ancestor of  $c$  and  $d$ , and  $\{a, b\} = \{c, d\}$  if the two pairs share the same least common ancestor in  $T$ . We can choose whether to include the minimal pairs  $\{a, a\}$ . Ng and Wormald [87] characterise when a given partial ordering on pairs equals the partial ordering of pairs of leaves in a tree.

Given a tree  $T$ , we have  $ab|c \in r(T)$  if and only if  $\{a, b\} \prec \{a, c\}$ . That's fine, but there is no collection of rooted triples corresponding to the statement  $\{a, b\} \prec \{c, d\}$ , nor a collection of nestings or  $n$ -taxon statements. As a consequence, our characterisation of compatibility goes only one way.

**Corollary 2.5** *Let  $S$  and  $T$  be rooted trees. If  $\mathcal{L}(S) \subseteq \mathcal{L}(T)$ , and  $\{a, b\} \prec \{c, d\}$  in  $S$  implies  $\{a, b\} \prec \{c, d\}$  in  $T$  then  $S \preceq T$ , but the converse is not true.*

*Proof*

Suppose that  $\mathcal{L}(S) \subseteq \mathcal{L}(T)$ , and  $\{a, b\} \prec \{c, d\}$  in  $S$  implies  $\{a, b\} \prec \{c, d\}$  in  $T$ . If  $ab|c \in r(S)$  then  $\{a, b\} \prec \{a, c\}$  in  $S$ , so  $\{a, b\} \prec \{a, c\}$  in  $T$  and  $ab|c \in r(T)$ . By Theorem 2.1 we have  $S \preceq T$ .

To show that the converse is not true consider the two trees in Figure 10. The tree  $T$  extends the tree  $S$ . But  $\{a, b\} \prec \{c, d\}$  in  $S$  even though we don't have  $\{a, b\} \prec \{c, d\}$  in  $T$ .  $\square$

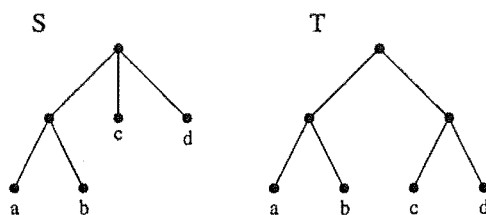


Figure 10 : Counterexample for proof of Corollary 2.5.

### 2.3.6 Direct Extensions

We mentioned earlier that some people use different versions of compatibility. One common variant excludes contractions. Under this version of compatibility, which we will call **direct extension**, a tree  $T$  directly extends a tree  $S$  if and only if  $S$  is an induced subtree of  $T$ , that is, if  $T$  is the same as  $S$  with extra vertices and edges added on.

For such extensions, fan trees and star trees become informative. Requiring that a tree directly extends a fan tree forces a multifurcation in the tree.

Given a rooted tree  $T$  recall that

$$f(T) = \{(abc) : T|_{\{a,b,c\}} = (abc)\}.$$

When  $T$  is unrooted we define

$$s(T) = \{(abcd) : T|_{\{a,b,c,d\}} = (abcd)\}$$

where  $(abcd)$  is the unrooted tree with no internal edges and leaf set  $\{a, b, c, d\}$ . It only seems fair to characterise direct extensions in all the ways that we characterised standard extensions, though we omit the proof.

**Theorem 2.6** *Let  $S$  and  $T$  be rooted trees. The following are equivalent:*

1.  $T$  is a direct extension of  $S$ .
2.  $S = T|_{\mathcal{L}(S)}$ .
3.  $r(S) = r(T)|_{\mathcal{L}(S)}$  ( $= \{ab|c \in r(T) : a, b, c \in \mathcal{L}(S)\}$ )
4.  $r(S) \subseteq r(T)$  and  $f(S) \subseteq f(T)$ .
5.  $\sigma(S) = \sigma(T)|_{\mathcal{L}(S)}$ .
6.  $A <_S B$  if and only if  $A <_T B$ , all  $A, B \subseteq \mathcal{L}(S)$ .
7.  $(A)B$  is an  $n$ -taxon statement of  $S$  if and only  $(A)B$  is an  $n$ -taxon statement of  $T$ , all  $A, B \subseteq \mathcal{L}(S)$ .
8.  $\{a, b\} < \{c, d\}$  in  $S$  if and only if  $\{a, b\} < \{c, d\}$ , for all  $a, b, c, d \in \mathcal{L}(S)$ .

Let  $S$  and  $T$  be unrooted trees. The following are equivalent:

1.  $T$  is a direct extension of  $S$ .
2.  $S = T|_{\mathcal{L}(S)}$ .
3.  $q(S) = q(T)|_{\mathcal{L}(S)}$  ( $= \{ab|cd \in q(T) : a, b, c, d \in \mathcal{L}(S)\}$ )
4.  $q(S) \subseteq q(T)$  and  $s(S) \subseteq s(T)$ .
5.  $\beta(S) = \beta(T)|_{\mathcal{L}(S)}$ .

## 2.4 When is a set of unrooted trees compatible?

Throughout the discussion so far we have been building up to the big question: when can the information extracted from a given collection of trees be combined and assembled into one larger tree? This problem appears in many places. Any attempt to incorporate the many existing phylogenies into one all encompassing phylogeny will come up against this problem, as will any ‘divide and conquer’ technique for large classifications.

In an ideal world there would exist a fast and efficient algorithm to solve this problem. Unfortunately, the general tree compatibility problem was shown to be NP-complete by Steel [101], making the existence of a fast and efficient tree compatibility algorithm unlikely.

We mentioned above (section 1.2, page 10) that NP-completeness is not necessarily cause for despair. Even when the general problem is NP-complete there are often special cases that have polynomial time solutions. This is true for tree compatibility. Sometimes it is easy. If a set of quartets contains two quartets on the same leaf set, then clearly the set is not compatible. If all the trees in the input set have the same leaf set, then compatibility can be determined in linear time [113]. Tree compatibility can also be determined in polynomial time if the input trees are rooted or all share a leaf in common (see below, section 2.5, and [6]).

### 2.4.1 Splits and Quartets

One of the earliest problems solved in this field was the problem of split compatibility. Given a set of splits  $\mathcal{S}$ , determine whether there is a tree  $T$  such that  $\mathcal{S} \subseteq \beta(T)$  (in



which case we say that  $\mathcal{S}$  is **compatible**). The solution comes in the form of two results that are now part of the folklore.

**Theorem 2.7** 1. Two splits  $A|B$  and  $C|D$  are compatible if and only if at least one of  $A \cap C$ ,  $A \cap D$ ,  $B \cap C$  or  $B \cap D$  is empty. [73, 28, 45]

2. A set of splits  $\mathcal{S}$  is compatible if and only if it is pairwise compatible. [28, 80]

We said earlier that it is easy to determine the compatibility of a set of trees with the same leaf set. The reason for this is that a set of trees  $\mathcal{T}$  on a leaf set  $L$  is compatible if and only if  $\cup_{T \in \mathcal{T}} \beta(T)$  is compatible [47]. We extend these folklore results by exploring links between splits and quartets.

Given a split  $A|B$  define

$$q(A|B) := \{ab|cd : a, b \in A, c, d \in B\},$$

the **quartet set of the split**  $A|B$ . It follows that  $A|B \in \beta(T)$  if and only if  $q(A|B) \subseteq q(T)$ . As well,  $q(T) = \cup_{A|B \in \beta(T)} q(A|B)$ . There is an attractive link between split compatibility and quartet compatibility.

**Lemma 2.8** 1. Two splits  $A|B$  and  $C|D$  are compatible if and only if  $q(A|B) \cup q(C|D)$  contains no quartet conflicts (i.e. two different quartets on the same leaf set).

2. A set of splits  $\mathcal{S}$  is compatible if and only if  $\cup_{A|B \in \mathcal{S}} q(A|B)$  contains no quartet conflicts.

*Proof*

1. Two splits  $A|B$  and  $C|D$  are incompatible if and only if  $\exists w, x, y, z$  such that  $w \in A \cap C$ ,  $x \in A \cap D$ ,  $y \in B \cap C$  and  $z \in B \cap D$  if and only if  $\exists w, x, y, z$  such that  $wx|yz \in q(A|B)$  and  $wy|xz \in q(C|D)$  if and only if  $q(A|B) \cup q(C|D)$  contains a quartet conflict.
2. A set of splits  $\mathcal{S}$  is compatible if and only if every pair of splits  $A|B$ ,  $C|D$  in  $\mathcal{S}$  are compatible if and only if  $\forall A|B, C|D \in \mathcal{S}$ , the set  $q(A|B) \cup q(C|D)$

contains no quartet conflicts if and only if  $\cup_{A|B \in \mathcal{S}} q(A|B)$  contains no quartet conflicts.  $\square$

We have reduced split compatibility down to quartet conflicts. But wait there's more. We say that a quartet  $ab|cd$  **contradicts** a split  $A|B$  if  $ac|bd \in q(A|B)$  or  $ad|bc \in q(A|B)$ . This gives rise to a useful set of quartets corresponding to a split  $A|B$ : the set of quartets that do *not* contradict it. Define

$$\hat{q}(A|B) := \{ab|cd : ac|bd \notin q(A|B) \text{ and } ad|bc \notin q(A|B)\}.$$

Note that there is at least one quartet in  $\hat{q}(A|B)$  for every set of four leaves. We can now recast Lemma 2.8.

**Lemma 2.9** *Two splits  $A|B$  and  $C|D$  are compatible if and only if  $q(A|B) \subseteq \hat{q}(C|D)$ .*

*Proof*

Suppose that  $A|B$  is compatible with  $C|D$ . Then  $q(A|B) \cup q(C|D)$  contains no quartet conflicts. Given any  $ab|cd \in q(A|B)$ , neither  $ac|bd$  nor  $ad|bc$  is in  $q(C|D)$ . Hence  $ab|cd$  does not contradict the split  $C|D$ , and  $ab|cd \in \hat{q}(C|D)$ .

Conversely if  $A|B$  and  $C|D$  are incompatible then there is  $w, x, y, z$  such that  $wx|yz \in q(A|B)$  and  $wy|xz \in q(C|D)$ . Hence  $wx|yz \notin \hat{q}(C|D)$  and  $q(A|B) \not\subseteq \hat{q}(C|D)$ .  $\square$

We extend this result to sets of splits.

**Theorem 2.10** *A set of splits  $\mathcal{S}$  is compatible if and only if*

$$\bigcup_{A|B \in \mathcal{S}} q(A|B) \subseteq \bigcap_{A|B \in \mathcal{S}} \hat{q}(A|B).$$

*Proof*

A set of splits  $\mathcal{S}$  is compatible if and only if  $A|B$  is compatible with  $C|D$  for all pairs of splits  $A|B$  and  $C|D$  in  $\mathcal{S}$ , if and only if  $q(A|B) \subseteq \hat{q}(C|D)$  for all  $A|B, C|D \in \mathcal{S}$  if and only if

$$\bigcup_{A|B \in \mathcal{S}} q(A|B) \subseteq \bigcap_{A|B \in \mathcal{S}} \hat{q}(A|B). \square$$

When does equality hold? Exactly when the splits correspond to the splits of a binary tree.

**Corollary 2.11** *A set of splits  $\mathcal{S}$  equals  $\beta(T)$  for some binary tree if and only if*

$$\bigcup_{A|B \in \mathcal{S}} q(A|B) = \bigcap_{A|B \in \mathcal{S}} \hat{q}(A|B).$$

*Proof*

Let  $T$  be a binary tree and put  $\mathcal{S} = \beta(T)$ . Since  $\mathcal{S}$  is compatible, Theorem 2.10 gives  $\bigcup_{A|B \in \mathcal{S}} q(A|B) \subseteq \bigcap_{A|B \in \mathcal{S}} \hat{q}(A|B)$ . Let  $ab|cd \in \bigcap_{A|B \in \mathcal{S}} \hat{q}(A|B)$ . Because  $T$  is binary,  $q(T)$  has a quartet for every set of four leaves and there is a split  $A|B \in \beta(T)$  such that either  $ab|cd \in q(A|B)$  or  $ac|bd \in q(A|B)$  or  $ad|bc \in q(A|B)$ . But  $ab|cd \in \hat{q}(A|B)$ , so  $ac|bd \notin q(A|B)$  and  $ad|bc \notin q(A|B)$ . Hence  $ab|cd \in q(A|B)$ , proving  $\bigcup_{A|B \in \mathcal{S}} q(A|B) = \bigcap_{A|B \in \mathcal{S}} \hat{q}(A|B)$ .

Conversely, suppose that  $\mathcal{S}$  is not the set of splits from some binary tree  $T$ . If  $\mathcal{S}$  is not compatible then  $\bigcup_{A|B \in \mathcal{S}} q(A|B) \not\subseteq \bigcap_{A|B \in \mathcal{S}} \hat{q}(A|B)$ . Suppose that  $\mathcal{S}$  is compatible, and let  $T$  be the non-binary tree such that  $\mathcal{S} = \beta(T)$ .

Since  $T$  is non-binary, there is  $a, b, c, d$  such that none of  $ab|cd, ac|bd$  or  $ad|bc$  are in  $q(T) = \bigcup_{A|B \in \mathcal{S}} q(A|B)$ . That means that none of  $ab|cd, ac|bd$  or  $ad|bc$  contradict any of the splits in  $\mathcal{S}$  and so  $ab|cd \in \hat{q}(A|B), \forall A|B \in \mathcal{S}$ . Therefore  $\bigcup_{A|B \in \mathcal{S}} q(A|B) \neq \bigcap_{A|B \in \mathcal{S}} \hat{q}(A|B)$ .  $\square$

One useful consequence of these results is that for any conflict free set of quartets  $Q$  there is a unique maximal tree  $T$  on the same leaf set such that  $q(T) \subseteq Q$ .

**Corollary 2.12** *Let  $Q$  be a set of quartets containing at most one of  $ab|cd, ac|bd, ad|bc$  for each set of four leaves  $a, b, c, d \in L$ . Put  $\mathcal{S} = \{A|B : q(A|B) \subseteq Q\}$ . Then  $\mathcal{S}$  is compatible.*

The tree  $T$  containing splits  $\mathcal{S}$  together with the trivial splits is clearly maximal in  $\{T : q(T) \subseteq Q, \mathcal{L}(T) = \mathcal{L}(Q)\}$  with respect to  $\preceq$ . We call it the **Bunemann (quartet) tree** because the Bunemann tree for a distance metric  $d$  equals the Bunemann quartet tree for  $Q = \{ab|cd : \min\{d_{ac} + d_{bd}, d_{ad} + d_{bc}\} > d_{ab} + d_{cd}\}$  [29].

### 2.4.2 Compatible Splits

We say that a split  $A|B$  is **compatible** with a set of quartets  $Q$  if  $Q \subseteq \hat{q}(A|B)$ , that is, if  $Q$  contains no quartets that contradict  $A|B$ . Therefore a consequence of

Corollary 2.11 is that a binary tree  $T$  is compatible with a set of quartets  $Q$  if and only if all the splits in  $\beta(T)$  are compatible with  $Q$ . This motivates us to study the set of all splits that are compatible with a set of quartets  $Q$ , denoted  $CS(Q)$ . We can break the tree compatibility problem into two parts:

1. Calculate the set of splits  $CS(Q) = \{A|B : Q \subseteq \hat{q}(A|B)\}$ .
2. Determine if there is a binary tree  $T$  such that  $\beta(T) \subseteq CS(Q)$ .

It turns out that there is an efficient algorithm for the second part (section 4.3, page 89), but there are several problems with carrying out the first part. After all, Quartet Compatibility is an NP-complete problem. The main difficulty is that there can be an exponentially large number of splits in  $CS(Q)$ : consider  $CS(\emptyset)$ . As well, we now show that the general problem of determining whether  $CS(Q)$  contains *any* non-trivial splits is NP-complete itself!

Recall that a split  $A|B$  of a leaf set  $L$  is non-trivial if  $|A| \neq 0, 1, |L| - 1, |L|$ .

### SPLIT-QUARTET COMPATIBILITY. (SQC)

INSTANCE: Set  $Q$  of quartets on a leaf set  $L$ .

QUESTION: Is there are non-trivial split compatible with  $Q$ ?

**Theorem 2.13** *SQC is NP-complete, even when  $Q$  contains at most one quartet on each set of four leaves.*

*Proof*

It is easy to see that  $SQC \in NP$  since a non-deterministic algorithm need only guess a non-trivial split of  $L$  and check in polynomial time whether that split is compatible with  $Q$ .

We transform 3SAT to SQC.

### 3-SATISFIABILITY (3SAT)

INSTANCE: Collection  $C = \{c_1, c_2, \dots, c_m\}$  of clauses on a finite set  $U$  of variables such that  $|c_i| = 3$  for  $1 \leq i \leq m$ .

QUESTION: Is there a truth assignment for  $U$  that satisfies all the clauses in  $C$ ?

Let  $U$  be a set of variables and  $C = \{c_1, c_2, \dots, c_m\}$  be a set of clauses making up an arbitrary instance of 3SAT. To construct the equivalent instance for SQC we use the leaf set:

$$L = \{T, T', F, F'\} \cup \{w_i : i = 1, \dots, m\} \cup U \cup \bar{U}$$

where  $\{T, T', F, F'\}$  and the  $w_i$ 's are new leaves, and  $\bar{U} = \{\bar{a} : a \in U\}$ . We construct the quartet set in four steps.

#### Step 1

$$\begin{aligned} Q_{TF} = & \{TT'|xy : x, y \in L - \{T, T'\}\} \\ & \cup \{FF'|xy : x, y \in L - \{F, F'\}\} \\ & \cup \{Tx|Fy : x \in L - \{F, F'\}, y \in L - \{T, T'\}\}. \end{aligned}$$

If  $A|B$  is any non-trivial split of  $L$  compatible with  $Q_{TF}$  and  $T \in A$  then  $T' \in A$ ,  $F \in B$  and  $F' \in B$ .

Suppose that  $A|B$  is compatible with  $Q_{TF}$ ,  $T \in A$  and  $T' \in B$ . Since  $A|B$  is non-trivial there is  $x, y \in L - \{T, T'\}$  such that  $x \in A$  and  $y \in B$ . But then  $A|B$  is not compatible with  $TT'|xy$ , which is in  $Q_{TF}$ .

In a similar way, both  $F$  and  $F'$  have to be on the same side of any such split  $A|B$ .

If  $T \in A$  and  $F \in A$  then also  $T', F' \in A$ . Choose  $x, y \in B$ . Then  $A|B$  is not compatible with the quartet  $Tx|Fy \in Q_{TF}$ .

#### Step 2

$$Q_{neg} = \{Ta|\bar{a}T', Fa|\bar{a}F' : a \in U\}$$

If  $A|B$  is any non-trivial split of  $L$  compatible with  $Q_{TF} \cup Q_{neg}$ , and  $T \in A$  then

$$x \in A \Leftrightarrow \bar{x} \in B.$$

If  $x$  and  $\bar{x}$  are both in  $A$  then  $A|B$  is not compatible with  $Fx|\bar{x}F'$ . If  $x$  and  $\bar{x}$  are both in  $B$  then  $A|B$  is not compatible with  $Tx|\bar{x}T'$ .

### Step 3

For  $1 \leq i \leq m$  write  $c_i = (x_i, y_i, z_i)$  where  $x_i, y_i, z_i \in U \cup \bar{U}$ . Construct

$$Q_i = \{Tx_i|w_iT', Tw_i|y_iF, Tw_i|z_iF, w_iy_i|z_iT\}.$$

If  $A|B$  is any non-trivial split compatible with  $Q_{TF} \cup Q_{neg} \cup Q_i$ , and  $T \in A$  then at least one of  $x_i, y_i$  or  $z_i$  is in  $A$ .

If  $y_i \in B$  and  $z_i \in B$  then  $w_i \in B$  since  $A|B$  is compatible with  $w_iy_i|z_iT$ . Hence  $x_i \in A$  because otherwise  $A|B$  would not be compatible with  $Tx_i|w_iT'$ .

### Step 4

$$Q = Q_{TF} \cup Q_{neg} \cup \left( \bigcup_{i=1}^m Q_i \right).$$

Clearly both  $L$  and  $Q$  can be constructed in polynomial time.

There is a non-trivial split  $A|B$  of  $L$  compatible with  $Q$  if and only if there is a truth assignment for  $U$  that satisfies  $C$ .

Let  $A|B$  be a non-trivial split of  $L$  compatible with  $Q$  such that  $T \in A$ . For  $x \in U$  put  $t(x) = TRUE$  if  $x \in A$ , and  $t(x) = FALSE$  if  $x \in B$ . We have already seen that if  $c_i = (x_i, y_i, z_i)$  is a clause then at least one of  $x_i, y_i, z_i$  is in  $A$ . Hence  $t$  satisfies  $c_i$ , and so  $t$  satisfies  $C$ .

Conversely, suppose that  $t : U \rightarrow \{TRUE, FALSE\}$  is a truth assignment for  $U$  that satisfies  $C$ . Put

$$\begin{aligned} A' &= \{T, T'\} \cup \{x : t(x) = TRUE\} \cup \{\bar{x} : t(x) = FALSE\} \\ B' &= \{F, F'\} \cup \{x : t(x) = FALSE\} \cup \{\bar{x} : t(x) = TRUE\}. \end{aligned}$$

Write  $c_i = (x_i, y_i, z_i)$  for  $1 \leq i \leq m$ , where  $x_i, y_i, z_i \in U \cup \bar{U}$ . Then put

$$\begin{aligned} A &= A' \cup \{w_i : y_i \in A' \text{ or } z_i \in A'\} \\ B &= B' \cup \{w_i : y_i \in B' \text{ and } z_i \in B'\} \end{aligned}$$

so that  $A|B$  is a split of  $L$ . Clearly  $A|B$  is compatible with  $Q_{TF} \cup Q_{neg}$ . Choose  $i$  from 1 to  $m$ . The assignment  $t$  satisfies  $c_i$  so at least one of  $x_i, y_i, z_i$  is assigned *TRUE*, and is hence in  $A$ .

If  $t(x_i) = \text{TRUE}$  and at least one of  $t(y_i)$  or  $t(z_i)$  (w.l.o.g.  $t(y_i)$ ) equals *TRUE*, then  $x_i, y_i, w_i \in A$ , so  $A|B$  is compatible with  $Q_i$ .

If  $t(x_i) = \text{TRUE}$  and  $t(y_i) = t(z_i) = \text{FALSE}$  then  $w_i, y_i, z_i \in B$  and  $x_i \in A$ , so  $A|B$  is compatible with  $Q_i$ .

If  $t(x_i) = \text{FALSE}$  then at least one of  $t(y_i)$  or  $t(z_i)$  (w.l.o.g.  $t(y_i)$ ) equals *TRUE* so  $w_i, y_i \in A$  and  $x_i \in B$ , so  $A|B$  is compatible with all four quartets in  $Q_i$ .

These are the only possibilities, and in all three cases  $A|B$  is compatible with  $Q_i$ . Hence  $A|B$  is a non-trivial split compatible with  $Q$ , completing the proof.  $\square$

Note that the problem is NP-complete in the *general* case. If  $Q$  is compatible then finding a non-trivial split compatible with  $Q$  is not difficult, since a set is compatible if and only if there is a binary tree that extends it and every binary tree contains a split  $A|B$  such that  $|A| = 2$ . These correspond to pendant pairs in the tree. Hence to find a non-trivial split we need only go through  $O(n^2)$  splits  $A|B$  with  $|A| = 2$ , checking each one to see if it is compatible with  $Q$ . This observation is incorporated into our tree building algorithm in the following section.

### 2.4.3 So how can we build trees?

Suppose that we are given a set of quartets that is not covered by the special cases listed at the beginning of section 2.4. We do not have any polynomial time algorithms for building a tree from quartets, so the set of quartets would have to be small. Here are three algorithms for determining compatibility of small to medium sized quartet sets.

The first method is about as inefficient as we could hope for: construct all possible binary trees on that leaf set and check to see if any of them extend the quartet set. The problem with this approach is that, for  $n$  leaves, there are  $1.3.5 \cdots (2n-5) = \frac{(2n-4)!}{(n-2)!2^{n-2}}$  different binary trees [92, 98].

The second method is well suited to determining compatibility by hand for quartet sets having up to 10 leaves, depending on the structure in the set. We conduct a recursive depth first search. First order the leaves  $L = \{a_1, \dots, a_n\}$ , put  $k = 3$  and

let  $T_3$  be the tree with leaves  $a_1, a_2, a_3$  joined to a central vertex. Pass  $k, T_k$  and the quartet set  $Q$  to the procedure CONSTRUCTTREE (Algorithm 1, below).

```

Procedure CONSTRUCTTREE( $k, T_k, Q$ )
1.  FOR all edges  $e$  in  $T_k$  DO
2.    Construct  $T_{k+1}$  from the tree  $T_k$  by subdividing  $e$  with a
        new internal vertex and appending  $a_{k+1}$  to this vertex.
3.    IF  $T_{k+1}$  extends  $Q|_{\{a_1, \dots, a_{k+1}\}}$  THEN
4.      IF  $k+1 = n$  THEN output this tree
5.      ELSE CONSTRUCTTREE( $k+1, T_{k+1}, Q$ )
6.    END(IF)
7.  END(FOR)
END.

```

Algorithm 1 : CONSTRUCTTREE

The algorithm will output every tree in  $\langle Q \rangle$ . If  $Q$  is incompatible then it will terminate without outputting a tree. In practice it is often possible to improve the efficiency of the algorithm by ordering the leaves carefully, or taking one of the quartets as the initial tree. The basic idea still applies.

Even if we stop as soon as we find a tree in  $\langle Q \rangle$ , this method can, in horrible cases, take as long as the crude ‘search all trees’ method, sometimes searching through almost all the possible trees.

In section 2.4.2 we hinted at a third method: construct  $CS(Q)$ , the set of splits that the quartets in  $Q$  do not contradict, and then run this set through the tree hunting algorithm of Chapter 4. For any given set of  $n$  leaves, there are  $2^{n-1}$  possible splits. By checking each split we can construct  $CS(Q)$  in  $O(|Q|2^{n-1}) = O(n^4 2^{n-1})$  time. Extracting a tree from these splits, if there is one, takes at most  $O(3^n)$  time by Theorem 4.5 in section 4.2.6.

This approach might give an improved worst case complexity, but is a bit heavy handed in practice. We describe a more refined version later, in Chapter 4 (section 4.6). For now we again put quartet compatibility aside and concentrate on rooted triples.



## 2.5 When is a set of rooted trees compatible?

Aho *et al.* [6] describe an algorithm that takes as input a collection of constraints of the form  $(a, b) < (c, d)$  where  $(a, b) < (c, d)$  means that the least common ancestor of  $a$  and  $b$  is a proper descendant of the least common ancestor of  $c$  and  $d$ . The algorithm quickly constructs a tree satisfying the constraints, if such a tree exists. They originally applied the algorithm to a problem in relational databases. They had also effectively solved the tree compatibility problem for rooted trees: given a collection of rooted trees determine whether there is a rooted tree that extends every tree in the collection.

Aho *et al.*'s algorithm has been extended and modified [33, 88, 66]. Constantinescu and Sankoff [33] present an algorithm SUPERB that takes a set of constraints and returns all of the binary trees that extend them, if any such trees exist. Ng and Wormald [88] give two tree construction algorithms: ONETREE and ALLTREES. These take both rooted triples and  $k$ -leaved fan trees as input, where a tree  $T$  is defined to be compatible with a fan tree  $S$  if  $S$  is an induced subtree of  $T$ . Hence the algorithms can determine whether there exists a tree that *directly* extends the trees in the input set. The algorithm ONETREE constructs a single such tree. ALLTREES lists all of them. Henzinger *et al.* [66] modify Aho *et al.*'s algorithm to make it run even faster.

Algorithm 2 is the same ONETREE [88] except that it has been restricted and simplified by removing the capability to handle fan trees. At each stage we construct the graph  $[R, S]$  as follows: take the leaves in  $S$  to be the vertices of the graph; add an edge between two vertices  $a$  and  $b$  if there are any triples in  $R$  of the form  $ab|c$  where  $a, b, c \in S$ ; label each edge  $(a, b)$  with the set of leaves  $\{x : ab|x \in R, x \in S\}$ .

For example, consider the graph  $[R, S]$  when

$$R = \{ab|c, be|d, be|c, af|g, ef|b, bf|a, bf|c, cd|a, cd|f, cg|b\}$$

and  $S = \{a, b, c, d, e, f\}$ . The triples  $af|g$  and  $cg|b$  are ignored, because  $g \notin S$ . The graph has six vertices and five edges. Note that the edges  $(b, f)$ ,  $(b, e)$  and  $(c, d)$  have multiple labels (Figure 11).

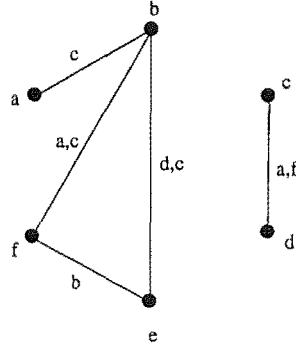


Figure 11 : The graph  $[R, S]$  for  $R = \{ab|c, be|d, be|c, af|g, ef|b, bf|a, bf|c, cd|a, cd|f, cg|b\}$  and  $S = \{a, b, c, d, e, f\}$ .

Once the graph is constructed we calculate its components and recurse. The algorithm takes as input a set of rooted triples  $R$  and a leaf set  $S = \{x_1, \dots, x_n\}$ .

Procedure ONETREE( $R, S$ )

1. IF  $n = 1$  THEN RETURN a single vertex labelled by  $x_1$ .
  2. IF  $n = 2$  THEN RETURN a tree with two leaves labelled  $x_1$  and  $x_2$ .
  3. Otherwise, construct  $[R, S]$  as described.
  4. IF  $[R, S]$  has only one component THEN RETURN 'No Tree'.
  5. FOR each component  $S_i$  of  $[R, S]$  DO
    6. IF ONETREE( $R, S_i$ ) returns a tree THEN call it  $T_i$  ELSE RETURN 'No Tree'.
  7. END(FOR)
  8. Construct a new tree  $T$  by connection the roots of the trees  $T_i$  to a new root  $r$ .
  9. RETURN  $T$ .
- END.

Algorithm 2 : ONETREE

The algorithm takes time  $O(mn)$  where  $m$  is the number of rooted triples in  $R$ .

### 2.5.1 Characterisation of Compatibility

The algorithm ONETREE takes a set of rooted triples and returns a tree if and only if the set is compatible. Looking at the algorithm, the only way that a tree wouldn't be returned is if the graph  $[R, S]$  is connected (step 4), where  $R$  and  $S$  are the original sets or those constructed during recursion. This gives rise to an important characterisation of compatibility for rooted triples, which we shall use frequently.

**Theorem 2.14** *A set of rooted triples  $R$  with leaf set  $L$  is compatible if and only if for each subset  $S \subseteq L$  with at least three elements, the graph  $[R, S]$  is disconnected.*

*Proof*

If  $R$  is compatible then there is a tree  $T$  such that  $R \subseteq r(T)$ . Choose  $S \subseteq L$  such that  $|S| \geq 3$ , and consider the subtree  $T|_S$ . The subtree has a greatest element in  $T$ , say  $M$ . Each child  $x$  of  $M$  determines a subset of  $S$  given by those leaves that are descendants of  $x$ . The collection of these subsets partitions  $S$  into two or more blocks (see Figure 12).

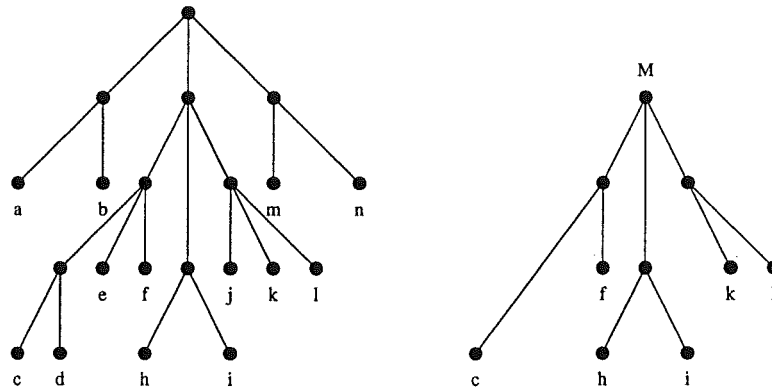


Figure 12 : Given the rooted tree on the left, take  $S = \{c, f, h, i, k, l\}$ . The induced subtree on the right has the corresponding partition  $\{\{c, f\}, \{h, i\}, \{k, l\}\}$ .

If  $a$  and  $b$  are elements from different blocks of this partition then there is no  $c \in S$  such that  $ab|c \in R$ . Therefore there is no edge in  $[R, S]$  between elements in different blocks of the partition, and so  $[R, S]$  is disconnected.

Conversely, suppose that  $R$  is incompatible. The algorithm **ONETREE** will return a null tree when applied to  $R$ . The algorithm acts recursively on different subsets  $S \subseteq L$ , and constructs the graph  $[R, S]$ . It only returns a null tree when for some leaf set  $S$  such that  $|S| \geq 3$ , this graph is connected.  $\square$

### 2.5.2 What is so special about the **ONETREE** tree?

We have so far only looked at the **ONETREE** algorithm as a means of determining compatibility. It is time to investigate the properties of the particular tree that the algorithm constructs. Given a set of rooted triples  $R$  the tree returned by **ONETREE** is called the *Aho et al. tree* for  $R$ . The question is whether the Aho et al. tree is representative of the information in  $R$  in a way that the other trees in  $\langle R \rangle$  are not.

The first surprising result is that Aho et al. tree is not minimal with respect to the number of vertices in a tree, or the number of rooted triples.

Consider the set of rooted triples  $R = \{bc|a, bd|a, ef|a, eg|a\}$ . The left hand tree in Figure 13 is the Aho et al. tree for  $R$ , the tree on the right is another tree extending  $R$ , and that tree contains fewer vertices and fewer rooted triples.

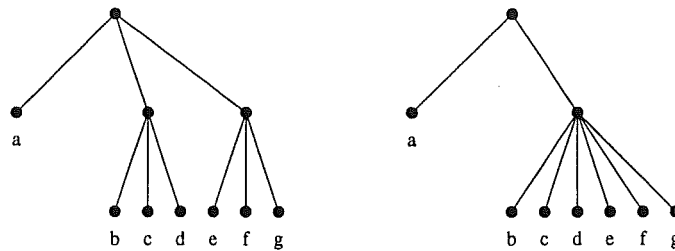


Figure 13 : Two trees that extend  $R = \{bc|a, bd|a, ef|a, eg|a\}$ .

The Aho et al. tree is closely linked to the well-known Adams consensus tree. We leave the definition of the Adams consensus tree and the proof of the following theorem to Chapter 6, sections 6.2.4 and 6.2.5.

**Theorem 2.15** *Given a set of rooted triples  $R$  on leaf set  $L$ , the Aho et al. tree for  $R$  equals the Adams Consensus tree for  $\langle R \rangle$ .*

It was proved in [2] that the Adams consensus tree contains the nesting information common to all the trees in the input set (see section 2.3.3 for a description of nesting). We use this property to show that the Aho *et al.* tree does satisfy another minimisation criteria.

**Theorem 2.16** *Let  $R$  be a compatible set of rooted triples and let  $AT$  be the Aho *et al.* tree for  $R$ . Let  $T$  be any other tree in  $\langle R \rangle$ . Choose any leaf  $a \in \mathcal{L}(T)$ . The distance from  $a$  to the root in  $T$  is greater than or equal to the distance from  $a$  to the root in  $AT$ , where all edges have equal edge weights.*

*Proof*

Consider the vertices on the path from  $a$  to the root in  $AT$ . Each vertex corresponds to a cluster of  $T$ , so we have a series of clusters  $C_0 \subset C_1 \subset C_2 \subset \dots \subset C_m$  where  $C_0 = \{a\}$ ,  $C_m = \mathcal{L}(T)$ , and  $m$  equals the distance from  $a$  to the root in  $AT$ .

Now  $AT$  is the Adams consensus tree for  $\langle R \rangle$  so for any  $T \in \langle R \rangle$  we have  $C_0 <_T C_1 <_T \dots <_T C_m$ . For each  $i = 1, \dots, m$  let  $C'_i$  denote the smallest cluster in  $\sigma(T)$  containing  $C_i$ . Then  $C_0 = C'_0 \subset C'_1 \subset \dots \subset C'_m = C_m$ . Each cluster  $C'_i$  corresponds to a different vertex on the path from  $a$  to the root of  $T$ . Hence the distance from  $a$  to the root is at least as much as the distance in the Aho tree  $AT$ .  $\square$

Note that the Aho *et al.* tree is not strictly minimal in this respect, a counter example being the trees in Figure 13 above.

The Aho *et al.* tree is also minimal with respect to the partial ordering  $\trianglelefteq$ . If  $T$  is the Aho *et al.* tree for  $R$  then there is no contraction of  $T$  that is also compatible with  $R$ . Note that given  $R$  there is not always a tree  $T_R$  such that  $T \in \langle R \rangle$  if and only if  $T_R \trianglelefteq T$ , an example being the rooted triple set in Figure 13. Nevertheless, if such a tree does exist then it will equal the Aho *et al.* tree for  $R$ , by virtue of the fact that the Aho *et al.* tree is minimal. It follows that, given  $T$ , the Aho *et al.* tree for  $r(T)$  equals  $T$ .

## 2.6 Some rooted tree problems that aren't easy

Rooted tree compatibility is easy, but there are minor variations on rooted tree compatibility that are NP-complete or NP-hard.

### 2.6.1 Maximum compatible subset of a rooted triple set

Suppose that we are given a set of rooted triples that is not compatible. Since we can't construct a tree on the whole set the best compromise is to construct a tree that extends the largest subset of the set. This problem, like the Maximum Compatible Subset of Splits Problem [39] is NP-Hard. The proof of this result was suggested by Tandy Warnow (personal communication).

#### MAXIMUM COMPATIBLE SUBSET OF ROOTED TRIPLES

INSTANCE Set  $R$  of rooted triples. Number  $0 \leq K \leq |R|$ .

QUESTION Is there a subset  $R'$  of  $R$  such that  $R'$  is compatible and  $|R'| \geq K$ ?

**Theorem 2.17** *MAXIMUM COMPATIBLE SUBSET OF ROOTED TRIPLES is NP-Complete.*

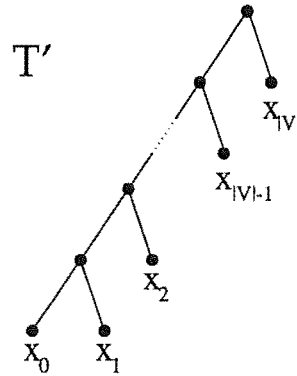
#### *Proof*

Clearly the problem is in NP, because we can just choose a subset of  $R$  and check in polynomial time if it is compatible and has more than  $K$  members.

We transform FEEDBACK ARC SET [55] into MAXIMUM COMPATIBLE SUBSET OF ROOTED TRIPLES.

Let  $G = (V, A); K \leq |A|$  make up an arbitrary instance of FEEDBACK ARC SET. Put  $L = V \cup \{x_0\}$ , where  $x_0$  is a new element. Construct  $R = \{ax_0|b : (a, b) \in A\}$ . We claim that a subset  $A' \subseteq A$  contains one arc from every directed cycle in  $G$  if and only if  $R - \{ax_0|b : (a, b) \in A'\}$  is compatible.

Suppose that  $A'$  contains one arc from every directed cycle in  $G$ . Then  $G' = (V, A - A')$  is acyclic so there exists an ordering  $x_1, x_2, \dots, x_{|V|}$  of  $V$  such that  $(x_i, x_j) \in A - A'$  implies  $i < j$ . Construct the rooted caterpillar tree  $T'$  as in Figure 14. This tree is compatible with  $R' = \{ax_0|b : (a, b) \in A - A'\}$  because  $x_i x_0 | x_j \in R'$  means  $(x_i, x_j) \in A - A'$  and so  $i < j$  and  $x_0 x_i | x_j \in r(T')$ .

Figure 14 : The caterpillar tree that extends  $R'$ .

Conversely suppose that  $R' = \{ax_0|b : (a, b) \in A - A'\}$  is compatible. There is  $T$  such that  $R' \subseteq r(T)$ . Now  $ab|c \in r(T)$  implies  $\{a, b\} \prec \{a, c\}$  in  $T$  (see section 2.3.5), so if there is a directed cycle  $(x_1, x_2, \dots, x_k, x_1)$  in  $G' = (V, A - A')$  then  $\{x_0x_1|x_2, x_0x_2|x_3, \dots, x_0x_k|x_1\} \subseteq R' \subseteq r(T)$  so  $\{x_0, x_1\} \prec \{x_0, x_2\} \prec \{x_0, x_3\} \prec \dots \prec \{x_0, x_k\} \prec \{x_0, x_1\}$ , a contradiction. Therefore  $A'$  does contain an arc from every directed cycle in  $G$ .

It follows that there is a subset  $A' \subseteq A$  with  $|A'| \leq K$  such that  $A'$  contains at least one arc from every directed cycle in  $G$  if and only if there is a subset  $R' \subseteq R$  with  $|R'| \geq |R| - K$  that is compatible. This completes the reduction and the proof.  $\square$

An extension of the MAXIMUM COMPATIBLE SUBSET OF ROOTED TRIPLES problem is to assign a weight to each of the rooted triples and then find the compatible subset with the greatest summed weight. This problem is clearly also NP-complete, because we could just choose a uniform weight and transform the original problem to it.

### 2.6.2 Forbidden Triples

Finding the maximum compatible subset of a set of splits is NP-hard [39] but we can quickly find a maximum compatible subset that corresponds to a binary tree (see Chapter 4). Does the same hold for the maximum compatible subset of rooted

triples problem? Given a set of rooted triples  $R$ , can we determine whether there is a binary tree  $T$  with leaf set  $\mathcal{L}(R)$  such that  $r(T) \subseteq R$ ? This is clearly equivalent to the following problem, which we call FORBIDDEN TRIPLES.

### FORBIDDEN TRIPLES

INSTANCE: A collection  $R$  of rooted triples whose leaf sets are subsets of a label set  $L$ .

QUESTION: Is there a binary tree  $T$  on leaf set  $L$  such that  $r(T) \cap R = \emptyset$ .

A related problem, FORBIDDEN SUBTREES, was shown to be NP-complete by Ng, Steel and Wormald [87].

### FORBIDDEN SUBTREES

INSTANCE: A collection  $S$  of rooted binary trees whose leaf sets are subsets of a label set  $L$ .

QUESTION: Is there a binary tree  $T$  with leaf set  $L$  having no subtree homeomorphic to a tree in  $S$ ?

Their proof involved a transformation from BETWEENNESS. However the set  $S$  that they constructed contained trees with four leaves, so their proof cannot be simply extended to prove the NP-completeness of FORBIDDEN TRIPLES. An implication of the proof in [87] is that determining whether there is a caterpillar tree  $T$  having no subtrees homeomorphic to a tree in  $S$  is NP-complete. This result does *not* hold if  $S$  contains only rooted triples, as Theorem 2.19 shows. For its proof we require:

**Lemma 2.18** *Let  $R$  be a set of rooted triples on leaf set  $L$  and let  $A|A'$  be any split of  $L$  such that for all  $ab|c \in R$  we have*

$$a, b \in A \implies c \in A \tag{1}$$

$$a, b \in A' \implies c \in A'. \tag{2}$$

*Then there exists a binary rooted tree  $T$  with leaf set  $L$  such that  $r(T) \cap R = \emptyset$  if and only if there exist binary trees  $T_1$  and  $T_2$  with leaf sets  $A$  and  $A'$  such that  $r(T_1) \cap R = \emptyset$  and  $r(T_2) \cap R = \emptyset$ .*



*Proof*

If there exists such a tree  $T$  then take  $T_1 = T|_A$  and  $T_2 = T|_{A'}$ . Conversely if there exist such trees  $T_1$  and  $T_2$  construct a tree  $T$  by attaching a root to the roots of  $T_1$  and  $T_2$ . Then

$$r(T) = r(T_1) \cup r(T_2) \cup \{ab|c : a, b \in A, c \in A'\} \cup \{ab|c : a, b \in A', c \in A\}$$

so  $r(T) \cap R = \emptyset$ .  $\square$

**Theorem 2.19** *Given a set of rooted triples  $R$  on leaf set  $L$  we can determine in  $O(|L|^2|R|)$  time whether there is a caterpillar tree  $T$  on leaf set  $L$  such that  $r(T) \cap R = \emptyset$ .*

*Proof*

Given a split  $A|B$  of  $L$  and a set of rooted triples  $R$  it takes  $O(|R|)$  time to check whether  $A|B$  is an allowed split, as defined in Lemma 2.18. Therefore all trivial splits can be checked in  $O(|L||R|)$  time. If none of these splits are allowed then there is no caterpillar tree, and if one of the splits is allowed, say  $x|L - x$  then recurse with leaf set  $L - x$  and rooted triple set  $R|_{L-x}$ . There are at most  $|L|$  levels of recursion, so this method takes  $O(|L|^2|R|)$  time.  $\square$

Despite the potential for a dynamical programming type algorithm based on Lemma 2.18 the general FORBIDDEN TRIPLES problem is NP-complete.

**Theorem 2.20** *FORBIDDEN TRIPLES is NP-complete.*

*Proof*

Clearly FORBIDDEN TRIPLES is in NP. We use a similar approach to the proof of Theorem 2.13 by describing a polynomial transformation from 3SAT to FORBIDDEN TRIPLES.

Let  $U$  be a set of variables and  $C = \{c_1, c_2, \dots, c_m\}$  be a set of clauses making up an arbitrary instance of 3SAT. To construct the equivalent instance for FORBIDDEN TRIPLES we use the leaf set:

$$L = \{T, F\} \cup \{w_i : i = 1, \dots, m\} \cup U \cup \bar{U}$$

where  $T, F$  and the  $w_i$ 's are new leaves, and  $\bar{U} = \{\bar{u} : u \in U\}$ .

Construct

$$R_{TF} = \{TF|x : x \in L - \{T, F\}\}$$

and

$$R_{neg} = \{u\bar{u}|T, u\bar{u}|F : u \in U\}.$$

For  $1 \leq i \leq m$  write  $c_i = (x_i, y_i, z_i)$  where  $x_i, y_i, z_i \in U \cup \bar{U}$ . Construct

$$R_i = \{x_i y_i | w_i, T x_i | w_i, T y_i | w_i, z_i w_i | T\}$$

and put

$$R = R_{TF} \cup R_{neg} \cup \bigcup_{i=1}^m R_i$$

Clearly  $R$  and  $L$  can be constructed in polynomial time. We claim that *there is a truth assignment for  $U$  satisfying  $C$  if and only if there is binary tree  $\tau$  with leaf set  $L$  such that  $r(\tau) \cap R = \emptyset$ .*

Let  $\tau$  be a tree satisfying these conditions. The tree  $\tau$  has two maximal subtrees. Let  $A$  and  $B$  be the leaf sets of these subtrees, where  $T \in A$ . Consider the truth assignment for  $U$  given by

$$t(x) = \begin{cases} TRUE & \text{if } x \in A \\ FALSE & \text{if } x \in B \end{cases}$$

The triples in  $R_{TF}$  force  $F \in B$ , and the triples  $R_{neg}$  force  $u$  and  $\bar{u}$  to be in different subtrees, all  $u \in U$ .

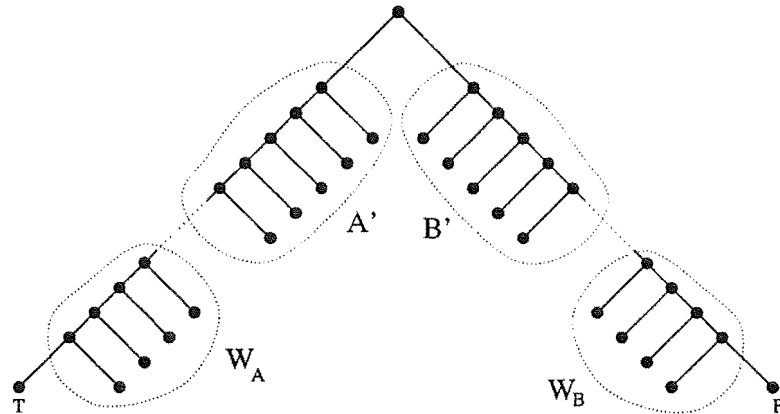
If  $(x_i, y_i, z_i) \in C$  and  $x_i, y_i \in B$  then  $w_i \in B$  because of the triple  $x_i y_i | w_i$ . Hence  $z_i \in A$  because of the triple  $z_i w_i | T$ . It follows that at least one of  $x_i, y_i, z_i$  are in  $A$ , so at least one element in the clause is assigned true.

Conversely, suppose that  $t : U \rightarrow \{TRUE, FALSE\}$  is a truth assignment for  $U$  that satisfies  $C$ . Put

$$\begin{aligned} A' &= \{T\} \cup \{x : t(x) = TRUE\} \cup \{\bar{x} : t(x) = FALSE\} \\ B' &= \{F\} \cup \{x : t(x) = FALSE\} \cup \{\bar{x} : t(x) = TRUE\}. \end{aligned}$$

Write  $c_i = (x_i, y_i, z_i)$  for  $1 \leq i \leq m$ , where  $x_i, y_i, z_i \in U \cup \bar{U}$ . Then put

$$\begin{aligned} W_A &= \{w_i : y_i \in A' \text{ or } z_i \in A'\} \\ W_B &= \{w_i : y_i \in B' \text{ and } z_i \in B'\} \end{aligned}$$

Figure 15 : The tree  $\tau$ 

Construct a binary tree  $\tau$  as in Figure 15. It is clear that  $r(\tau) \cap R_{TF} = \emptyset$  and  $r(\tau) \cap R_{neg} = \emptyset$ . For each  $i$  the fact that  $r(\tau) \cap R_i = \emptyset$  follows from our choice of whether to place  $w_i$  in  $W_A$  or  $W_B$ . Hence  $r(\tau) \cup R = \emptyset$ .  $\square$

The proof of Theorem 2.20 also provides us with an alternative proof for the NP-completeness of FORBIDDEN SUBTREES.

### 2.6.3 Maximum Resolved Tree

We mentioned (above, page 35) that Aho *et al.*'s algorithm has been used to determine if there exists a tree that directly extends the trees in the input set. Given a set of fans  $F$  and a set of rooted triples  $R$  we can determine if there is a tree  $T$  such that  $R \subseteq r(T)$  and  $F \subseteq f(T)$ . The question is, what is the most resolved tree  $T$  such that  $R \subseteq r(T)$  and  $F \subseteq f(T)$ ? If  $F = \emptyset$  then clearly the most resolved tree is any binary tree in  $\langle R \rangle$ . But when  $F \neq \emptyset$  the problem becomes more difficult, even when  $R = \emptyset$ .

#### MOST RESOLVED COMPATIBLE TREE

INSTANCE Set  $R$  of rooted triples and set  $F$  of fans. Number  $K \geq 0$ .

**QUESTION** Is there a tree  $T$  with  $K$  or more internal edges such that  $R \subseteq r(T)$  and  $F \subseteq f(T)$ .

**Theorem 2.21** *MOST RESOLVED COMPATIBLE TREE is NP-complete, even when  $R = \emptyset$  and there is a leaf common to all fans.*

*Proof*

It is easy to see that MOST RESOLVED COMPATIBLE TREE is in NP. We transform GRAPH K-COLORABILITY into MOST RESOLVED COMPATIBLE TREE.

Let  $G = (V, E)$ ;  $K > 0$  make up an arbitrary instance of GRAPH K-COLORABILITY. We can assume that  $G$  is connected. We construct a set of fans  $F$  on the leaf set  $V \cup \{x\}$ , where  $x$  is a new vertex. Initially put  $F = \emptyset$ . For every edge  $(a, b)$  in  $G$  add the fan  $(a, b, x)$  to  $F$ . We claim that  $G$  is  $K$ -colourable if and only if there is a tree strictly compatible with  $F$  with  $n - K$  internal edges, where  $n = |V|$ .

Let  $f : V \rightarrow \{1, 2, \dots, K\}$  be a  $K$ -colouration of  $G$ . For each  $i = 1, 2, \dots, K$  construct an arbitrary binary tree  $T_i$  with leaf set  $f^{-1}(i)$ . Assemble the tree  $T$  by appending the roots of the trees  $T_i$  to a new root  $r$ , and then appending the vertex  $x$  to this root (Figure 16). Given any fan  $(a, b, x)$  in  $F$ , the vertices  $a$  and  $b$  are adjacent in  $G$  so receive different colours. Hence  $a, b$  and  $x$  are in different subtrees descending off the root  $r$ , and  $T$  directly extends  $(a, b, x)$ . The number of internal edges in each subtree  $T_i$ , including the edge connecting it with the root, equals  $|f^{-1}(i)| - 1$ . Hence the number of internal edges in  $T$  equals  $n - K$ .

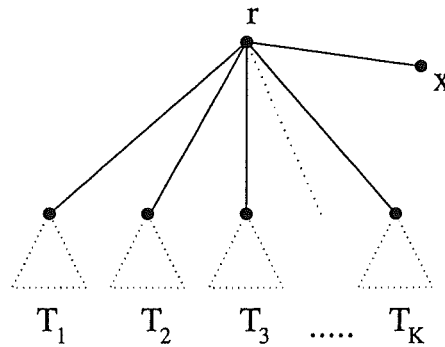


Figure 16 : The tree  $T$ , constructed from a  $K$ -colouring of  $G$ .

Conversely, let  $T$  be a tree with  $n - K$  internal edges that directly extends  $F$ . If there is a vertex  $a$  such that  $a$  and  $x$  are both contained in a subtree branching off the root of  $T$ , then every leaf  $b$  adjacent to  $a$  in  $G$  is also in the same subtree, since  $(a, b, x) \in F$ . It follows that all the vertices of  $G$  are contained in that subtree, since  $G$  is connected. A contradiction. Hence  $x$  must be in a maximal subtree by itself.

Let  $T_1, T_2, \dots, T_J$  be the remaining subtrees branching off the root of  $T$ . We let  $n_i$  equal the number of leaves in  $T_i$  and let  $e_i$  equal the number of internal edges in  $T_i$ , including the edge connecting the root of  $T_i$  to the root of  $T$ . Then  $e_i \leq n_i - 1$  so we have

$$n - K = \sum_{i=1}^J e_i \leq \sum_{i=1}^J (n_i - 1) = n - J$$

and  $J \leq K$ .

Given any two leaves  $a$  and  $b$  in a subtree  $T_i$ , the fan  $(a, b, x)$  is not in  $F$ , so there is no edge between  $a$  and  $b$  in  $G$ . Let  $f$  be the function from  $V$  to  $\{1, 2, \dots, J\}$  such that  $f(a) = i$  if and only if  $a$  is a leaf in  $T_i$ . Then  $f$  is a  $J$ -colouring of  $G$  and can easily be modified to give a  $K$  colouring of  $G$ , since  $K \geq J$ .  $\square$



## Chapter 3

# Building Trees - Inference Rules

### 3.1 Introduction

Phylogenetics is often described as tree *reconstruction*, the assumption being that there exists a ‘true tree’ that researchers are trying to recover, in whole or in part. This assumption alone has significant ramifications, even if we do not know anything about the true tree.

For example suppose that a particular biologist is convinced that a collection of trees is ‘true’, that is, the ‘true tree’ is an extension of the trees in the collection. One necessary condition for this belief to be logically consistent is that the trees are compatible. But we can often infer more. Suppose that every tree extending a given collection also extends an additional tree. Then the statement that the initial set is true implies that the additional tree is also true. This is the basic idea behind inference rules.

We showed in the previous chapter (section 2.3.1) that any question relating to the compatibility of unrooted trees can be converted into a question about quartets, likewise for rooted trees and rooted triples. Therefore we can, without loss of generality, discuss compatibility rules, or inference rules, in terms of sets of quartets or rooted triples, rather than in terms of general rooted and unrooted trees.

To study inference rules we introduce the closure operator and closed sets, those sets to which no additional quartets (or rooted triples) can be added by application of inference rules. We characterise closed sets, and discuss several situations where they arise.

Focusing on rooted triple rules, we show that the graphical characterisation of compatibility in the previous chapter (Theorem 2.14, page 37) leads to a graphical characterisation of closed sets and several elegant properties of closed sets of rooted triples. In particular, we show that there exist rules of every order that cannot be reduced to repeated application of lower order rules and that no finite set of rules will suffice to determine compatibility.

Many of the elegant properties of closed sets of rooted triples do not hold for closed sets of quartets. It is possible, however, to extend the result about irreducible rooted triple rules to show that there exist quartet rules of every order that cannot be reduced to repeated application of lower order rules.

## 3.2 Introduction to inference rules and closed sets

A **quartet rule** or **inference rule** is a statement of the form “If  $Q \subseteq q(T)$  then  $ab|cd \in q(T)$ ” and is denoted  $Q \vdash ab|cd$ . Hence  $Q \vdash ab|cd$  is true if every tree that extends  $Q$  also extends  $ab|cd$ , that is

$$ab|cd \in \bigcap_{T \in \langle Q \rangle} q(T).$$

Consider some simple examples. If we take the quartets  $Q = \{ab|cd, ab|ce\}$  and run one of the tree-building algorithms of section 2.4.3, then we see that there are exactly four trees in the span of  $Q$  (Figure 17). All of these trees extend the quartet  $ab|de$ . Hence  $\{ab|cd, ab|ce\} \vdash ab|de$ .

There is only one tree in the span of  $Q_2 = \{ab|cd, ac|de\}$ , (Figure 17) and this tree has quartet set  $\{ab|cd, ab|ce, ab|de, ac|de, bc|de\}$ . Hence  $Q_2 \vdash ab|ce$  and  $Q_2 \vdash ab|de$  and  $Q_2 \vdash bc|de$ .



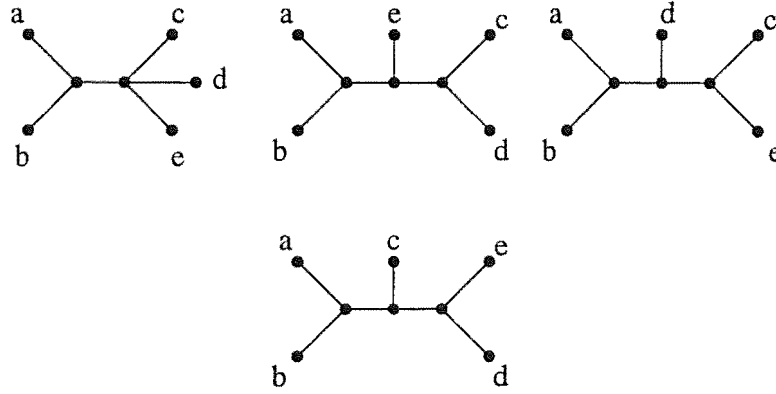


Figure 17 : The four trees in the span of  $\{ab|cd, ab|ce\}$ . The bottom tree is the only tree in the span of  $\{ab|cd, ac|de\}$ .

There are six trees in the span of  $Q_3 = \{ab|cd, ab|ef, ce|df\}$ , (Figure 18), and  $ab|df$  is a quartet in all of them. Hence  $Q_3 \vdash ab|df$ .

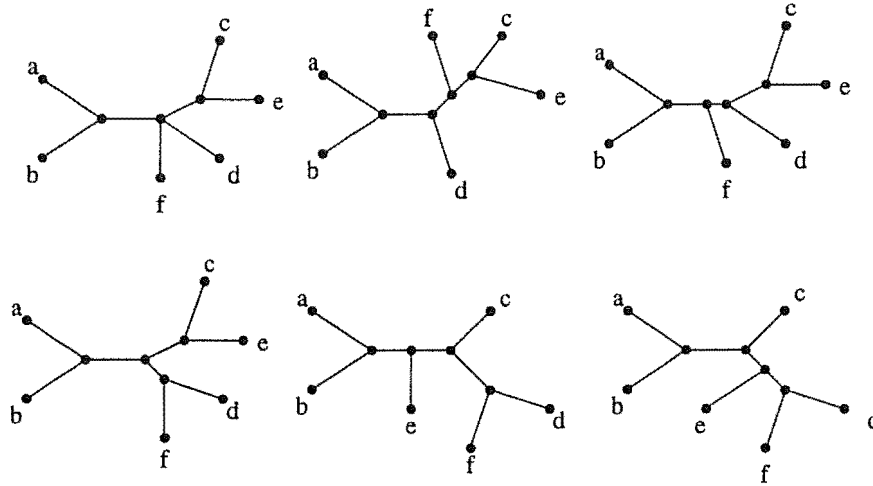


Figure 18 : The six trees in the span of  $\{ab|cd, ab|ef, ce|df\}$ .

The **order** of a quartet rule  $Q \vdash q$  equals the cardinality of  $Q$ . The rules  $\{ab|cd, ab|ce\} \vdash ab|de$  and  $\{ab|cd, ac|de\} \vdash ab|ce$  have order two. Dekker [40] calls these rules the **dyadic rules** and shows that they are the only fundamentally distinct rules of order two.

A rule is **irreducible** if it cannot be derived from repeated application of rules with lesser order. The rule  $\{ab|cd, ab|ef, ce|df\} \vdash ab|df$  is an irreducible rule of order three [40], but the rule  $\{bc|df, ae|df, be|cf\} \vdash ab|df$  can be reduced to the dyadic rules  $\{bc|df, be|cf\} \vdash be|df$  and  $\{be|df, ae|df\} \vdash ab|df$  (see Section 3.4.3).

Given a compatible set of quartets define

$$\overline{Q} = \bigcap_{T \in \langle Q \rangle} q(T).$$

Thus  $Q \vdash ab|cd$  is a rule if and only if  $ab|cd \in \overline{Q}$ . The set  $\overline{Q}$  is called the **closure** of  $Q$ . A set  $Q$  is **closed** if every rule  $Q \vdash ab|cd$  implies that  $ab|cd \in Q$ .

So far we have only defined rules and closure and closed sets for quartets. The definitions can be easily extended to rooted triples. Given a compatible set of rooted triples  $R$ , we have that  $R \vdash ab|c$  is a rule if and only if  $ab|c \in \bigcap_{T \in \langle R \rangle} r(T)$ . The order of  $R \vdash ab|c$  equals  $|R|$ , and the closure of  $R$  is given by

$$\overline{R} = \bigcap_{T \in \langle R \rangle} r(T).$$

There are two fundamentally distinct rooted triple rules of order two:  $\{ab|d, ac|d\} \vdash bc|d$  and  $\{ab|c, bc|d\} \vdash ab|d$ . These are called the rooted triple dyadic rules.

### 3.3 Properties of Closed Sets

We present a number of basic properties of closed sets and the closure operator, all of which follow immediately from the definitions of closure and closed sets.

**Theorem 3.1** *Let  $X, Y$  both be two compatible sets of quartets (or rooted triples).*

1.  $\overline{X}$  is the minimal closed set containing  $X$ .
2.  $\overline{\overline{X}} = \overline{X}$ .
3. If  $X \subseteq Y$  then  $\overline{X} \subseteq \overline{Y}$ .
4.  $X$  is closed if and only if  $X = \overline{X}$ .
5. If  $X$  and  $Y$  are closed sets then  $X \cap Y$  is also closed.
6.  $T$  extends  $X$  if and only if  $T$  extends  $\overline{X}$ .

7.  $\langle X \rangle = \langle Y \rangle$  if and only if  $\overline{X} = \overline{Y}$ .

8.  $\overline{X \cup Y} = \overline{X} \cup \overline{Y}$ .

The definition of closure suggests a link between closed sets and quartet sets of trees. In fact, the quartet sets of binary trees are the maximal closed sets, and all other closed sets can be written as the intersection of them.

**Theorem 3.2**  *$X$  is closed if and only if  $X = q(T_1) \cap q(T_2) \cap \dots \cap q(T_k)$  for some trees  $T_1, T_2, \dots, T_k$ . Furthermore we can assume that  $T_1, T_2, \dots, T_k$  are binary.*

*Proof*

If  $T$  is a tree then  $\overline{q(T)} = q(T)$  so  $q(T)$  is closed. If  $X = q(T_1) \cap q(T_2) \cap \dots \cap q(T_k)$  for some trees  $T_1, T_2, \dots, T_k$  then  $X$  is closed, by Theorem 3.1 (5).

Conversely if  $X$  is closed then  $X = \overline{X}$  which is, by definition, the intersection of the quartet sets of all the trees that extend  $X$ . We can restrict our attention to binary trees because the quartet set  $q(T)$  of any non-binary tree  $T$  equals the intersection of the quartet sets of the binary trees that extend  $T$ .  $\square$

**Theorem 3.3** *If  $X$  and  $Y$  are both compatible sets of quartets and  $\mathcal{L}(X) \cap \mathcal{L}(Y) = \emptyset$  then  $X \cup Y$  is compatible and  $\overline{X \cup Y} = \overline{X} \cup \overline{Y}$ .*

*Proof*

First assume that there are unrooted trees  $T_1$  and  $T_2$  with disjoint leaf sets such that  $X = q(T_1)$  and  $Y = q(T_2)$ . We can combine  $T_1$  and  $T_2$  into a single tree by identifying an internal vertex of  $T_1$  with an internal vertex of  $T_2$ . Any tree constructed in this way extends  $q(T_1) \cup q(T_2)$ , so  $X \cup Y$  is compatible. Each different pair of internal vertices gives rise to a different tree. Let  $\mathcal{T}$  be the collection of all these trees. Then

$$q(T_1) \cup q(T_2) = \bigcap_{T \in \mathcal{T}} q(T)$$

so  $X \cup Y (= q(T_1) \cup q(T_2))$  is closed by Theorem 3.2.

Suppose that  $X$  and  $Y$  are any two compatible sets such that  $\mathcal{L}(X) \cap \mathcal{L}(Y) = \emptyset$ . By the definition of closure,

$$\overline{X} = \bigcap_{T_1 \in \langle X \rangle} q(T_1) \text{ and } \overline{Y} = \bigcap_{T_2 \in \langle Y \rangle} q(T_2).$$

Hence

$$\begin{aligned}\overline{X \cup Y} &= \left( \bigcap_{T_1 \in \langle X \rangle} q(T_1) \right) \cup \left( \bigcap_{T_2 \in \langle Y \rangle} q(T_2) \right) \\ &= \bigcap_{T_1 \in \langle X \rangle, T_2 \in \langle Y \rangle} (q(T_1) \cup q(T_2))\end{aligned}$$

which is closed by the first part and Theorem 3.1 (5). Now  $\overline{X \cup Y} \subseteq \overline{X \cup Y}$ , and  $\overline{X \cup Y}$  is the minimal closed set containing  $X \cup Y$ . Since  $\overline{X \cup Y}$  is closed, it follows that  $\overline{X \cup Y} = \overline{X \cup Y}$ , as required.  $\square$

Our definition of closed sets fulfils the criterion of a general *closure property* of Birkoff and MacLane [21]. The maximal closed sets are exactly the quartet sets (rooted triple sets) of binary trees. The meet of any two closed sets equals their intersection, and the join of two closed sets, if it exists, equals the closure of their union. Given a closed set  $Q$ , the set of closed subsets of  $Q$  form a complete lattice [21].

The closure operator for quartets and rooted triples defined here is not a matroid closure operator. For example,  $ab|ce \in \overline{\{ab|cd, ac|de\}} - \overline{\{ab|cd\}}$  but  $ac|de \notin \overline{\{ab|cd, ab|ce\}} - \overline{\{ab|cd\}}$ . Nevertheless we can still pilfer ideas and concepts from matroid theory. As an example, consider the concept of an independent set.

A set  $Q$  of quartets (rooted triples) is **independent** if there is no proper subset  $Q' \subset Q$  such that  $\overline{Q'} = \overline{Q}$ .

**Theorem 3.4** 1. If  $X$  is independent and  $Y \subseteq X$  then  $Y$  is independent.

2.  $X$  is independent if and only if  $\overline{A} = \overline{B}$  implies  $A = B$  for all  $A, B \subseteq X$ .

*Proof*

(1) Suppose that  $X$  is independent, but  $Y \subset X$  is not independent. There is  $Z \subset Y$  such that  $\overline{Z} = \overline{Y}$ . But then

$$\overline{(X - Y) \cup Z} = \overline{(X - Y) \cup \overline{Z}} = \overline{(X - Y) \cup \overline{Y}} \supset \overline{(X - Y) \cup Y} = \overline{X}$$

even though  $(X - Y) \cup Z \subset X$ .

(2) If  $X$  is independent,  $A, B \subseteq X$  and  $\overline{A} = \overline{B}$  then

$$\overline{A \cup B} = \overline{A \cup \overline{B}} = \overline{A \cup \overline{A}} = \overline{A}.$$

But  $A \cup B \subseteq X$  so  $A \cup B$  is independent. Therefore  $A = A \cup B$ . By symmetry  $B = A \cup B$ , so  $A = B$ .

If  $X$  is not independent then there is  $A \subset X$  such that  $\overline{A} = \overline{X}$ . Put  $B = X$ , then  $\overline{A} = \overline{B}$  but  $A \neq B$ .  $\square$

A set  $Q$  is **fully closed** if  $X \subseteq Q$  implies  $X$  closed. For example, if no two quartets in  $Q$  share any leaves, then  $Q$  is closed by repeated application of Theorem 3.3. Hence every subset of  $Q$  is closed and  $Q$  is fully closed.

We give extensive treatment to fully closed sets later (section 3.4.1), but for now we will settle with an attractive characterisation in terms of independent sets.

**Theorem 3.5** *A set  $X$  of quartets (rooted triples) is fully closed if and only if  $X$  is independent and closed.*

*Proof*

Suppose that  $X$  is fully closed, then  $\overline{A} = A$  and  $\overline{B} = B$  for all  $A, B \subseteq X$ , so  $X$  is independent by Theorem 3.4.

Conversely, suppose that  $X$  is independent and closed. Given any  $Y \subseteq X$  we have that  $\overline{Y} \subset X$  since  $Y$  is closed. Then  $\overline{Y} = \overline{\overline{Y}}$  so  $Y = \overline{Y}$  and  $Y$  is closed. Hence  $X$  is fully closed.  $\square$

### 3.3.1 Applications of closed sets

#### Closed sets in consensus

Let  $\mathcal{T}$  be a collection of trees  $T_1, T_2, \dots, T_k$  and let  $Q = q(T_1) \cap q(T_2) \cap \dots \cap q(T_k)$ . The set of quartets  $Q$  is often taken to be the consensus information shared by all the trees in  $\mathcal{T}$  (See below, Chapter 6). By Theorem 3.2 any such set is closed. As well, if  $S$  is any consensus tree such that  $S \trianglelefteq T_i$ ,  $i = 1, \dots, k$ , then  $q(S) \subseteq Q$ . Note that other consensus methods are in use. Some consensus trees, like the Adams consensus tree for rooted trees [2], preserve more information than is contained in the intersection the rooted triple sets.

### Closed sets to represent tree sets

Let  $Q$  be a set of quartets or trees and let  $n = |\mathcal{L}(Q)|$ . The number of trees that extend  $Q$  can be exponentially large with respect to  $n$ , so it is often impractical to list every possible tree. Instead we could use the closed set  $\overline{Q}$  to represent the set of possible trees. The set  $\overline{Q}$  contains exactly those quartets that can be directly deduced from  $Q$ .

### Defining a tree

A natural question to ask is “When does a set of quartets define a tree?” That is, when does the span of a set of quartets  $Q$  contain exactly one tree? An answer is provided by the closure operator. If  $\langle X \rangle$  consists of just one tree  $T$  then  $\overline{X} = q(T)$ . Conversely if  $\overline{X} = q(T)$  for some binary tree  $T$  then

$$\langle X \rangle = \langle \overline{X} \rangle = \langle q(T) \rangle = \{T\}.$$

Therefore a set of quartets  $X$  determines a tree  $T$  if and only if  $T$  is binary and  $\overline{X} = q(T)$ . Of course the fact that we do not have a polynomial time algorithm for calculating the closure of a set means that this observation is primarily of theoretical interest. It is still unknown whether the problem is actually NP-complete (see Appendix A). Note that the equivalent problem for rooted trees is easy to solve [101].

### Quartets from multi-state characters

Phylogenetic information is often given by sets of characters. Each character gives a partition of the set of species, that is, a partition of the leaf set. The states of a character correspond to the blocks in the partition.

There is a corresponding notion of compatibility with partitions. A tree is compatible with a partition if edges can be removed from the tree to give subtrees with leaf sets equal to the blocks of the partition. The problem of determining whether such a tree exists for a given collection of partitions is called the Perfect Phylogeny Problem. It is NP-complete in general [101, 22] though can be solved in polynomial time if the number of partitions is bounded [82], or the number of states in each character, that is, the number of blocks in each partition, is bounded [4, 67].

Given a partition  $A_1|A_2|\dots|A_k$  define the set of quartets

$$q(A_1|A_2|\dots|A_k) := \{wx|yz : w, x \in A_i; y, z \in A_j; i \neq j\}.$$

It can be shown that a tree  $T$  is compatible with the partition  $A_1|A_2|\dots|A_k$  if and only if  $q(A_1|A_2|\dots|A_k) \subseteq q(T)$ . Given a set of quartets  $Q$ ,  $q(A_1|A_2|\dots|A_k) \subseteq \overline{Q}$  if and only if every tree in  $\langle Q \rangle$  extends the partition  $A_1|A_2|\dots|A_k$ . The inference rules of [40] involving partitions can therefore be reduced to inference rules involving quartets, giving additional motivation for studying sets of quartets.

We prove that for any partition  $A_1|A_2|\dots|A_k$ , the set of quartets  $q(A_1|A_2|\dots|A_k)$  is closed. In order to do so we consider quartet sets of graphs that are not necessarily trees.

**Lemma 3.6** *Let  $G$  be any connected graph and let  $L$  be a set of labelled vertices in  $G$ .*

*Define*

$$q(G) := \left\{ ab|cd : \begin{array}{l} a, b, c \text{ and } d \text{ are distinct elements of } L \\ \text{no path from } a \text{ to } b \text{ intersects a path from } c \text{ to } d \end{array} \right\}$$

*Then  $q(G)$  is compatible and closed.*

*Proof*

We can assume that every vertex in  $G$  is on a path between two elements of  $L$  since removing these vertices does not change the set  $q(G)$ .

Consider first the case when  $G$  is acyclic. Suppose that  $G$  has an internal vertex labelled  $a$ . If we attach a new leaf adjacent to this vertex and transfer the label  $a$  from the internal vertex to the leaf  $a$  then  $q(G)$  will not change. Repeat this procedure until all labelled vertices of  $G$  are leaves. If we delete those vertices that have only two remaining adjacent vertices and identify their incident edges then we obtain a phylogenetic tree  $T$  with  $q(T) = q(G)$ . Hence  $q(G)$  is compatible and closed.

Suppose now that  $G$  is not acyclic. Let  $\tau$  be a spanning tree of  $G$ . The subgraph  $\tau$  is acyclic so  $q(\tau)$  is the quartet set of some phylogenetic tree. If  $ab|cd \in q(G)$  then no path from  $a$  to  $b$  intersects a path from  $c$  to  $d$  in  $G$  and because  $\tau$  is a subgraph of  $G$ , the same applies for  $\tau$ . Hence  $q(G) \subseteq q(\tau)$  and  $q(G)$  is compatible.

Let  $\mathcal{T}$  be the collection of spanning trees of  $G$ . We will show that

$$q(G) = \bigcap_{\tau \in \mathcal{T}} q(\tau)$$

proving that  $q(G)$  is closed (Theorem 3.2). If  $ab|cd \notin q(G)$  then there is a path  $P_1$  from  $a$  to  $b$  that intersects a path  $P_2$  from  $c$  to  $d$ . Let  $x$  be the *first* vertex on the path  $P_2$  that is also on the path  $P_1$ . Let  $y$  be the *last* vertex on the path  $P_2$  that is also on the path  $P_1$ . Construct the subgraph of  $G$  containing all of  $P_1$ , the part of  $P_2$  going from  $c$  to  $x$  and the part of  $P_2$  going from  $y$  to  $d$ . This subgraph is an independent set of the graph matroid so can be extended to a spanning tree  $\tau$  of  $G$  for which  $ab|cd \notin q(\tau)$  [114]. Hence  $ab|cd \notin \bigcap_{\tau \in \mathcal{T}} q(\tau)$ .  $\square$

Unfortunately not every compatible closed set equals  $q(G)$  for some graph  $G$ , a counterexample being the set  $\{ab|cd, ab|ef\}$ .

**Theorem 3.7** *If  $A_1|A_2|\dots|A_k$  is any partition then  $q(A_1|A_2|\dots|A_k)$  is compatible and closed.*

*Proof*

Consider the graph  $G$  of Figure 19. Clearly  $q(G) = q(A_1|A_2|\dots|A_k)$  so, by Lemma 3.6, the set of quartets is compatible and closed.  $\square$

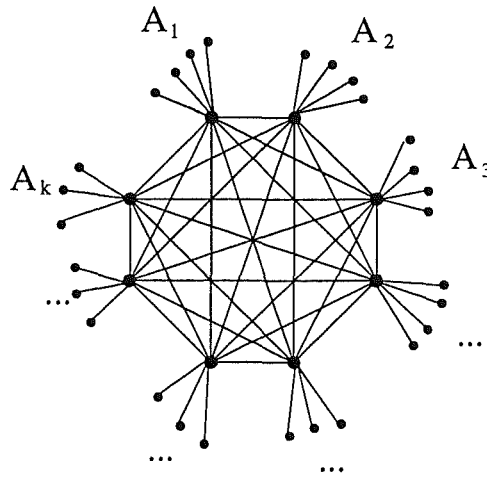


Figure 19 : The graph  $G$  with the same quartet set as  $A_1|A_2|\dots|A_k$ .



## 3.4 Closed sets and Rooted Triples

The graphical characterisation of compatibility introduced in Chapter 2 (Theorem 2.14, page 37) has proved to be extremely useful for exploring properties of rooted triple sets. We can not only characterise compatibility, but rooted triple inference rules and closure as well. The characterisations lead to several properties of closed sets of rooted triples, properties that do not hold, or at least do not appear to hold, for closed sets of quartets.

We also obtain proofs for two longstanding conjectures by Dekker [40]: that there are irreducible rooted triple rules of any order and that no finite collection of rooted triple rules suffices to determine compatibility.

### 3.4.1 Graphical characterisation of closure

Consider the rule  $R \vdash ab|c$ , where  $R$  is an arbitrary set of compatible rooted triples. Since every tree that extends  $R$  also extends  $ab|c$  the sets  $R \cup \{ac|b\}$  and  $R \cup \{bc|a\}$  must both be incompatible sets. Even though  $R$  is itself compatible, if we add the triples  $bc|a$  or  $ac|b$  then the set is incompatible. We characterise when this situation occurs, and then describe rooted triple rules in terms of the graphical representation.

**Theorem 3.8** *If  $R$  is a compatible set of rooted triples and  $R \cup \{ab|c\}$  is incompatible then  $\{a, b, c\} \subseteq \mathcal{L}(R)$  and there is a leaf set  $S$  with  $\{a, b, c\} \subseteq S$  such that the graph  $[R, S]$  has exactly two components, one containing  $a$  and the other containing  $b$ .*

*Proof*

Let  $T$  be a rooted tree that extends  $R$ . If any of  $a$ ,  $b$  or  $c$  are not in  $\mathcal{L}(R)$  then we can add these extra leaves to  $T$  in such a way to give a tree that extends  $R \cup \{ab|c\}$ , contradicting the incompatibility of  $R \cup \{ab|c\}$ .

By Theorem 2.14 (above, page 37) there is a set  $S$  with  $|S| \geq 3$  such that the graph  $[R \cup \{ab|c\}, S]$  is connected. The graph  $[R \cup \{ab|c\}, S]$  is the same as the graph  $[R, S]$  with one extra edge connecting  $a$  and  $b$  and labelled by  $c$ . Hence  $[R, S]$  has at most two components and since  $R$  is compatible, the graph must have exactly two components. Adding the edge  $(a, b)$  gives a connected graph, so  $a$  and  $b$  must be in different components of  $[R, S]$ .  $\square$

**Corollary 3.9**  $R \vdash ab|c$  if and only if there is  $S$  such that  $[R, S]$  has exactly two components, one containing  $a$  and  $b$ , and the other containing  $c$ .

*Proof*

If  $R \vdash ab|c$  then  $R \cup bc|a$  is incompatible even though  $R$  is compatible. Hence there is  $S$  such that  $\{a, b, c\} \subseteq S$  and  $[R, S]$  has two components, one containing  $b$  and the other containing  $c$ . If  $a$  is in the component containing  $c$  then  $[R \cup \{ab|c\}, S]$  has only one component, making  $R \cup \{ab|c\}$  incompatible by Theorem 2.14 (page 37), and contradicting the fact that  $R \vdash ab|c$ . Hence  $a$  and  $b$  are in the same component, and  $c$  is in the other component.

Conversely, if there is  $S$  such that  $[R, S]$  has two components, one containing  $a$  and  $b$  and the other containing  $c$ , then  $[R \cup \{bc|a\}, S]$  and  $[R \cup \{ac|b\}, S]$  are both connected, so  $R \cup \{bc|a\}$  and  $R \cup \{ac|b\}$  are both incompatible. Since  $R$  itself is compatible,  $R \cup \{ab|c\}$  is compatible and  $R \vdash ab|c$ .  $\square$

Now that we have characterised rooted triple rules we can characterise closed sets of rooted triples.

**Theorem 3.10** Let  $R$  be a compatible set of rooted triples.  $R$  is closed if and only if for each set  $S$ :  $|S| \geq 3$  for which  $[R, S]$  has exactly two components, these components are cliques and the label set of each edge contains every label in the other component.

*Proof*

Suppose that  $R$  is closed. Let  $S$  be a subset of  $\mathcal{L}(R)$  such that  $[R, S]$  has two components. Choose any  $a$  and  $b$  in one component and any  $c$  in the other component. By Corollary 3.9,  $R \vdash ab|c$ . Since  $R$  is closed,  $ab|c \in R$  so there is an edge between  $a$  and  $b$  with  $c$  in its label set. The result follows.

Conversely suppose that  $R$  is not closed. There is a rooted triple  $ab|c$  not contained in  $R$ , even though  $R \vdash ab|c$ . There is  $S \subseteq \mathcal{L}(R)$ ,  $|S| \geq 3$ , such that  $[R, S]$  has two components, with  $a$  and  $b$  in one component and  $c$  in the other. But since  $ab|c$  is not in  $R$ , the edge from  $a$  to  $b$  does not have  $c$  in its label set.  $\square$

One implication of this result is that if  $R$  is compatible and  $[R, S]$  has more than two components for all  $S$  :  $|S| > 3$  then  $R$  must be closed by default. We can say

more than that. Any such set  $R$  is fully closed, which means that  $R$  is closed and every subset of  $R$  is closed (section 3.3, above). This condition is both necessary and sufficient for fully closed sets of rooted triples.

**Theorem 3.11** *A compatible set  $R$  is fully closed if and only if for all  $S \subseteq \mathcal{L}(R)$  with  $|S| \geq 4$ , the graph  $[R, S]$  has at least three components.*

*Proof*

Suppose that for all  $S \subseteq \mathcal{L}(R)$  with  $|S| \geq 4$  the graph  $[R, S]$  has at least three components. Let  $R'$  be a subset of  $R$ . Choose any  $S \subseteq \mathcal{L}(R)$  such that  $|S| \geq 3$ . If  $|S| > 3$  then  $[R', S]$  has at least three components, since  $[R', S]$  is a subgraph of  $[R, S]$ . Alternatively, if  $|S| = 3$  then either there is no triple in  $R'$  with leaf set  $S$ , in which case  $[R', S]$  contains three components, or there is one triple in  $R'$  with leaves  $S$ , in which case  $[R', S]$  has one edge, labelled by the vertex in the second component. By Theorem 3.10,  $R'$  is closed. We conclude that  $R$  is fully closed.

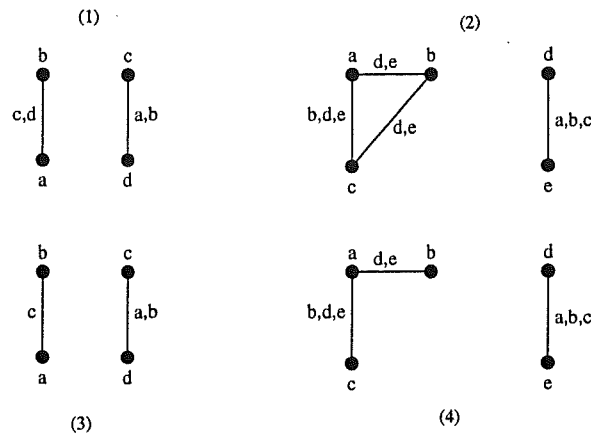


Figure 20 : If  $R$  is closed and  $[R, S]$  has two components, then either both components have two vertices (1), or one component has three *or more* vertices (2). In (3) we have removed the triple  $ab|d$  from  $R$  giving a set that is not closed. In (4) we have removed both  $bc|d$  and  $bc|e$ , giving a subset of  $R$  that is not closed.

Conversely, let  $R$  be fully closed. Consider any subset  $S \subseteq \mathcal{L}(R)$  with  $|S| > 3$ . The set  $R$  is compatible, so by Theorem 2.14 on page 37, the graph  $[R, S]$  has at

least two components. Suppose that  $[R, S]$  has only two components. Either both components have exactly two vertices, or one component has *at least* three vertices. In the first case both components have exactly one edge and this edge is labelled by the two vertices in the other component (Figure 20 (1)). Removing one triple from  $R$  that has leaves in  $S$  will remove one of the labels from one of the edges, giving a set that is not closed by Theorem 3.10 (Figure 20 (3)). In the second case (Figure 20 (2)), removing an edge will still leave a graph with two components that corresponds to a subset of  $R$  that is not closed (Figure 20 (4)). In either case,  $R$  is not fully closed.  $\square$

### 3.4.2 Properties of closed sets of rooted triples

A **conflict** in a set of quartets or rooted triples is a pair of different quartets or rooted triples with the same leaves. Clearly if a set of quartets or rooted triples contains a conflict then the set is incompatible. Dekker [40] observed that if sets of rules are applied to a set of quartets and a conflict is derived, then the set of quartets is incompatible. This is also true for sets of rooted triples. We prove, in the rooted triple case, that if we apply all possible rooted triple rules then the converse of Dekker's observation is also true (Theorem 3.12 (2) below).

- Theorem 3.12**
1. *If  $R$  is a closed set of rooted triples containing no triple with the leaves  $\{a, b, c\}$  then  $R \cup \{ab|c\}$ ,  $R \cup \{ac|b\}$  and  $R \cup \{bc|a\}$  are all compatible.*
  2. *If all possible rooted triple rules are applied to the compatible subsets of a set  $R$  of rooted triples then a conflict is derived if and only if the set  $R$  is incompatible.*
  3. *If  $R$  is a set of at least three rooted triples and every proper subset of  $R$  is closed (and therefore compatible), then  $R$  is compatible.*

*Proof*

(1) Suppose that one of  $R \cup \{ab|c\}$ ,  $R \cup \{ac|b\}$  and  $R \cup \{bc|a\}$  is incompatible, say  $R \cup \{ab|c\}$ . By Theorem 3.8, there is  $S : |S| \geq 3$  such that  $[R, S]$  has exactly two components, with  $a$  and  $b$  in different components.

If  $c$  is in the same component as  $a$  then  $[R \cup bc|a, S]$  is connected so  $R \cup bc|a$  is incompatible (Figure 21). Since  $R \cup ab|c$  is also incompatible we have, by elimination, that every tree that extends  $R$  also extends  $ac|b$ . That is,  $ac|b \in \overline{R}$ . Similarly, if  $c$  is in the same component as  $b$  then  $bc|a \in \overline{R}$ . In either case we obtain a contradiction.

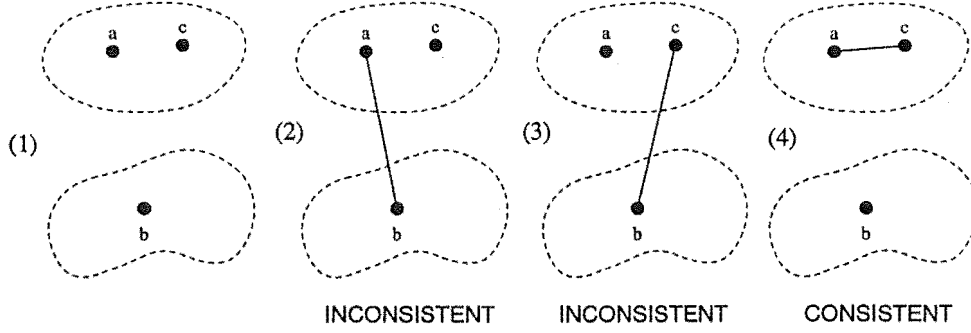


Figure 21 : The components of  $[R, S]$ . The dotted lines indicate the components of  $[R, S]$  (1). If we add an edge between  $a$  and  $b$ , (2), or between  $c$  and  $b$ , (3), then we get a connected graph.

Hence the only triple with these leaves that is compatible with  $R$  is  $ac|b$  (4).

(2) If  $R$  is compatible then  $\overline{R}$  is also compatible, so applying all possible rules to  $R$  will give a compatible set that contains no conflicts.

Conversely, suppose that  $R$  is incompatible. Let  $R_M$  be a maximal compatible subset of  $R$  and choose  $ab|c$  in  $R - R_M$ . Then  $\overline{R_M} \cup \{ab|c\}$  is incompatible, so by (1) either  $ac|b \in \overline{R_M}$  in which case  $R_M \vdash ac|b$ , or  $bc|a \in \overline{R_M}$  and  $R_M \vdash bc|a$ . In both cases we derive a triple conflicting with  $ab|c$ .

(3) Let  $ab|c \in R$ . Then  $R - \{ab|c\}$  is compatible and closed. Every subset of  $R$  is compatible, so  $R$  contains at most one of  $ab|c, ac|b, bc|a$ . Hence  $ac|b \notin R - \{ab|c\}$  and  $bc|a \notin R - \{ab|c\}$  so  $R = (R - \{ab|c\}) \cup \{ab|c\}$  is compatible by part (1).  $\square$

### 3.4.3 Irreducible Rooted Triple Rules

Contemplate the rule  $\{bc|d, ae|d, be|c\} \vdash be|d$ . It has order three, but it can be inferred directly from the dyadic rule  $\{bc|d, be|c\} \vdash be|d$ . The same applies to the

rule  $\{bc|d, ae|d, be|c\} \vdash ab|d$  except that this time we apply the dyadic rule twice: first on  $\{bc|d, be|c\}$  to obtain the rooted triple  $be|d$  as above, and then on  $\{be|d, ae|d\}$  to obtain  $ab|d$ . It was this kind of reduction that our definition of irreducibility describes (above, 50). Recall that an inference rule is irreducible if it cannot be derived by repeated application of lower order rules.

Dekker found irreducible quartet rules of orders three, four and five, and then conjectured that there exist irreducible rules of every order. We prove this result, first for rooted triple rules and then later (below, section 3.5.3) for quartet rules. The result is intriguing: there is an unexpected level of complexity stemming from such a simply defined concept.

Our proof is constructive. Given any  $n > 3$  we describe a set  $R$  consisting of  $n$  rooted triples that is compatible and *not* closed, even though every subset of  $R$  is closed. Because  $R$  is not closed there is  $ab|c \notin R$  such that  $R \vdash ab|c$ . Because every subset of  $R$  is closed applying any rule of order  $n - 1$  or less will not add any extra triples. Hence the rule  $R \vdash ab|c$  can not be derived through repeated application of lower order rules.

We begin with an example of a fully closed set that forms the basis of the construction. Note the heavy dependence on graphical characterisation.

**Theorem 3.13** *Let  $A = \{a_1, a_2, \dots, a_p\}$ ,  $B = \{b_1, b_2, \dots, b_q\}$ , and  $C = \{c_1, c_2, \dots, c_r\}$ , be disjoint sets of leaves. Let  $R$  be any set of rooted triples each of which is of the form  $a_i a_j | b_k$  or  $b_i b_j | c_k$  or  $c_i c_j | a_k$ , and which has the further property that for each  $z \in A \cup B \cup C$  there is at most one triple in  $R$  of the form  $xy|z$ . Then  $R$  is compatible and fully closed.*

*Proof*

The tree  $T$  in Figure 22 extends  $R$ , so  $R$  is compatible. We will use Theorem 3.11 to show that  $R$  is fully closed. Let  $S$  be a subset of  $\mathcal{L}(R)$  with  $|S| \geq 4$ .

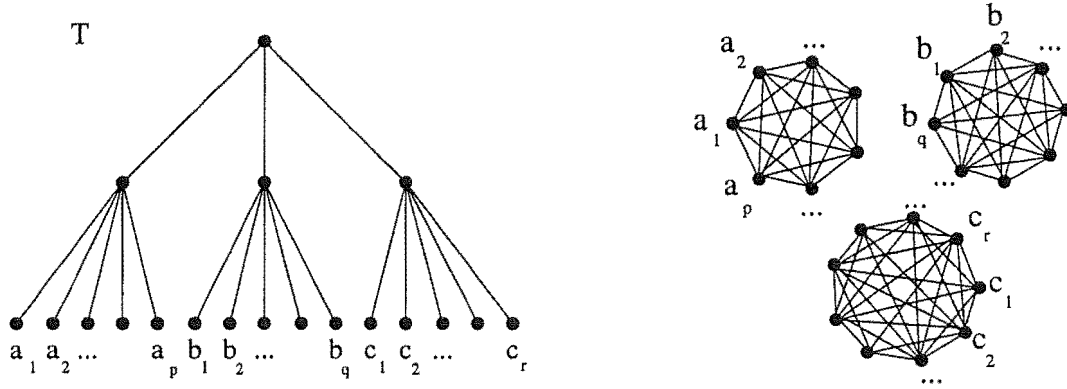


Figure 22 : The tree  $T$  on the left extends the set  $R$  from Theorem 3.13, so  $R$  is compatible.

On the right is the graph  $[r(T), \mathcal{L}(T)]$ , consisting of three disjoint cliques on  $p$ ,  $q$  and  $r$  vertices respectively. Note that  $[R, S]$  is a subgraph of  $[r(T), \mathcal{L}(T)]$  for any  $S$ .

Consider three cases:

Case 1:  $S$  contains at least one element from each of  $A$ ,  $B$  and  $C$ .

Now  $R \subseteq r(T)$  so  $[R, S]$  is a subgraph of  $[r(T), \mathcal{L}(T)]$  (Figure 22). The elements in  $A$ ,  $B$  and  $C$  must be in different components of  $[R, S]$ . Therefore  $[R, S]$  has at least three components.

Case 2 :  $S$  intersects exactly two of  $A$ ,  $B$  and  $C$ .

By symmetry, we can assume without loss of generality that  $S$  is contained in  $A \cup B$ . The graph  $[R, S]$  has at least two components because  $R$  is compatible (Theorem 2.14, page 37). Suppose that  $[R, S]$  has only two components. As in Case 1, the elements in  $A$  and the elements in  $B$  are contained in different components of  $[R, S]$ . Since  $[R, S]$  has only two components, the vertices in  $S \cap A$  are connected and the vertices of  $S \cap B$  are connected.

If there is more than one vertex in  $S \cap B$  then there is an edge in this component. However any such edge would be labelled by a vertex from  $C$ , giving a contradiction. On the other hand if there is only one vertex in the  $S \cap B$  component, then there must be at least three vertices in the  $S \cap A$  component, since  $|S| \geq 4$ . Hence there are at least two distinct edges in the  $S \cap A$  component. We required  $R$  to have the property that for each  $z$  in  $\mathcal{L}(R)$  there is at most one triple in  $R$  of the form  $xy|z$ . Each of these edges in  $S \cap A$  is therefore labelled by a different element of  $S \cap B$ , a contradiction. We conclude that  $[R, S]$  has at least three components.

Case 3:  $S$  is a subset of  $A$ ,  $B$  or  $C$ .

Without loss of generality, assume that  $S \subseteq A$ . If  $[R, S]$  has less than three components, then there must be an edge in  $[R, S]$ , simply because  $S$  has at least four elements. However all edges connecting vertices in  $A$  are labelled by vertices in  $B$ , so  $S$  must contain an element of  $B$  as well, a contradiction.

In all three cases,  $[R, S]$  has three components so  $R$  is fully closed by Theorem 3.11.

□

Back to the construction. It turns out that we actually construct not one but three sets of rooted triples.

$$\begin{aligned} R_0 := & \{a_1a_2|b_1, a_2a_3|b_2, \dots, a_ma_{m+1}|b_m, \\ & b_1b_2|c_1, b_2b_3|c_2, \dots, b_{m-1}b_m|c_{m-1}, \\ & c_1c_2|a_1, c_2c_3|a_2, \dots, c_mc_{m+1}|a_m, \\ & a_{m+1}b_1|c_{m+1}\}, \end{aligned} \quad m \geq 1 \quad (3)$$

$$\begin{aligned} R_1 := & \{a_1a_2|b_2, a_2a_3|b_3, \dots, a_ma_{m+1}|b_{m+1}, \\ & b_1b_2|c_1, b_2b_3|c_2, \dots, b_mb_{m+1}|c_m, \\ & c_1c_2|a_1, c_2c_3|a_2, \dots, c_mc_{m+1}|a_m, \\ & a_{m+1}b_1|c_{m+1}\}, \end{aligned} \quad m \geq 1 \quad (4)$$

$$\begin{aligned} R_2 := & \{a_1a_2|b_1, a_2a_3|b_2, \dots, a_ma_{m+1}|b_m, \\ & b_1b_2|c_1, b_2b_3|c_2, \dots, b_{m-1}b_m|c_{m-1}, \\ & c_1c_2|a_1, c_2c_3|a_2, \dots, c_{m-1}c_m|a_{m-1}, \\ & a_{m+1}b_1|c_m\}, \end{aligned} \quad m \geq 2 \quad (5)$$

**Lemma 3.14** *For each  $i \in \{0, 1, 2\}$ , the set  $R_i$  is compatible. Furthermore, if  $S$  is a proper subset of  $\mathcal{L}(R_i)$ ,  $i \in \{0, 1, 2\}$ , and  $|S| \geq 4$  then  $[R_i, S]$  has at least three components.*

*Proof*

We prove the case of  $i = 1$ . The remaining cases are proved in a similar way. Let  $R = R_1$ . The tree in Figure 23 extends  $R$  so  $R$  is compatible. Let  $A = \{a_1, a_2, \dots, a_{m+1}\}$ ,  $B = \{b_1, b_2, \dots, b_{m+1}\}$ ,  $C = \{c_1, c_2, \dots, c_{m+1}\}$ .



By Theorem 3.13, the set  $R - \{a_{m+1}b_1|c_{m+1}\}$  is fully closed, so by Theorem 3.11, the graph  $[R - \{a_{m+1}b_1|c_{m+1}\}, S]$  has at least three components. If one of  $a_{m+1}, b_1$  or  $c_{m+1}$  is not in  $S$  then  $[R, S]$  is the same graph as  $[R - \{a_{m+1}b_1|c_{m+1}\}, S]$  and so also has three components.

Assume that  $\{a_{m+1}, b_1, c_{m+1}\} \subset S$ . Since  $S \subseteq \mathcal{L}(R)$ , the graph  $[R, S]$  is a subgraph of  $[R, \mathcal{L}(R)]$  (see Figure 23). The elements of  $S \cap C$  and the elements of  $S \cap (A \cup B)$  are in different components in  $[R, \mathcal{L}(R)]$ , so they are in different components of  $[R, S]$ . Because  $S$  contains elements of both  $A \cup B$  and  $C$ , the graph  $[R, S]$  has at least two components. Suppose that  $[R, S]$  has only two components. Then all of the vertices in  $S \cap (A \cup B)$  are connected, and all the vertices in  $S \cap C$  are connected.

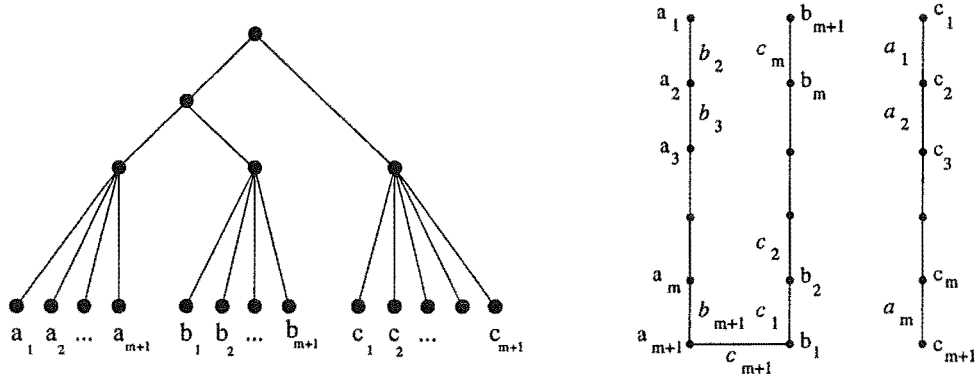


Figure 23 : The tree on the left extends  $R$ , so that  $R$  is compatible. On the right is the graph  $[R, \mathcal{L}(R)]$  and associated edge labelling.

There are at least four elements in  $S$ , so there is at least one additional element  $x$  in  $S$  other than  $a_{m+1}, b_1$  and  $c_{m+1}$ . Consider the cases of  $x \in A$ ,  $x \in B$  and  $x \in C$ .

Case 1:  $x \in A$

Let  $x = a_i$ . The vertices  $a_i$  and  $a_{m+1}$  are in the same component of  $[R, S]$ , so there is a path in  $[R, S]$  going from  $a_i$  to  $a_{m+1}$ . Now the only path in  $[R, \mathcal{L}(R)]$  (Figure 23) from  $a_i$  to  $a_{m+1}$  passes through  $a_i, a_{i+1}, \dots, a_m$ , and  $a_{m+1}$ . Since  $[R, S]$  is a subgraph of  $[R, \mathcal{L}(R)]$ , the only possible path from  $a_i$  to  $a_{m+1}$  in  $[R, S]$  passes through these same vertices. Therefore  $a_i, a_{i+1}, \dots, a_m, a_{m+1}$  and the labels of the edges connecting them in  $[R, S]$  are also in  $S$ . In particular the edge connecting  $a_m$

and  $a_{m+1}$  is in  $[R, S]$ , so  $b_{m+1} \in S$ .

But  $b_{m+1}$  is in the same component of  $[R, S]$  as  $b_1$  and  $a_i$ . Therefore there is a path in  $[R, S]$  from  $b_{m+1}$  to  $b_1$ . Referring to the graph  $[R, \mathcal{L}(R)]$  we observe that the only path from  $b_{m+1}$  to  $b_1$  in  $[R, \mathcal{L}(R)]$ , and hence in the subgraph  $[R, S]$ , passes through every vertex of  $B$ . Therefore all the vertices in  $B$  are also in  $S$ , as well as the labels of the edges connecting them in  $[R, S]$ . In particular the edge connecting  $b_1$  and  $b_2$  is in  $[R, S]$ , so  $c_1 \in S$ .

But  $c_1$  is in the same component of  $[R, S]$  as  $c_{m+1}$ . Therefore there is a path in  $[R, S]$  from  $c_1$  to  $c_{m+1}$ . All the vertices in  $C$  are also in  $S$ , as well as the labels of the edges connecting them in  $[R, S]$ . In particular the edge connecting  $c_1$  and  $c_2$  is in  $[R, S]$ , so  $a_1 \in S$ . Therefore there is a path from  $a_1$  to  $a_{m+1}$  in  $[R, S]$  and all the vertices in  $A$  are also in  $S$ . We have shown that  $S = \mathcal{L}(R)$ , giving a contradiction.

#### Case 2: $x \in C$

Let  $x = c_i$ . The vertices  $c_i$  and  $c_{m+1}$  are in the same component of  $[R, S]$ . Therefore there is a path in  $[R, S]$  from  $c_i$  to  $c_{m+1}$ . This is only possible if  $c_i, c_{i+1}, \dots, c_m$  and the labels of the edges connecting them in  $[R, S]$  are also in  $S$ . In particular the edge connecting  $c_m$  and  $c_{m+1}$  is in  $[R, S]$ , so  $a_m \in S$ . Hence  $a_{m+1}$  is not the only element of  $A$  in  $S$ . Referring to the first case we obtain a contradiction.

#### Case 3: $x \in B$

Let  $x = b_i$ . The vertices  $b_i$  and  $b_1$  are in the same component of  $[R, S]$ . Therefore there is a path in  $[R, S]$  from  $b_1$  to  $b_i$ . This is only possible if  $b_2, b_3, \dots, b_i - 1$  and the labels of the edges connecting them in  $[R, S]$  are also in  $S$ . In particular the edge connecting  $b_1$  and  $b_2$  is in  $[R, S]$ , so  $c_1 \in S$ . Hence  $c_{m+1}$  is not the only element of  $C$  in  $S$ . Referring to the second case we obtain a contradiction.  $\square$

*At last* we are ready to prove the result. We define the set  $R(n)$  for  $n > 3$ , as follows:

If  $n \equiv 0 \pmod{3}$  then put  $m = n/3$  and  $R(n) = R_0$ .

If  $n \equiv 1 \pmod{3}$  then put  $m = (n - 1)/3$  and  $R(n) = R_1$ .

If  $n \equiv 2 \pmod{3}$  then put  $m = (n + 1)/3$  and  $R(n) = R_2$ .

The sets  $R_0, R_1$  and  $R_2$  are given by equations 3, 4 and 5 on page 66.

In all three cases,  $R(n)$  has  $n$  triples.

**Theorem 3.15** *Given any  $n > 3$  there is a compatible set of  $n$  rooted triples that is not closed even though every proper subset is closed. Thus there is a rooted triple rule of order  $n$  that cannot be derived by repeated application of rules of order less than  $n$ .*

*Proof*

Put  $R = R(n)$ , then  $R$  is compatible by the preceding Lemma. We will use Theorem 3.11 to prove that every proper subset  $R'$  of  $R$  is fully closed. Choose  $R' \subseteq R$  and  $S \subseteq \mathcal{L}(R')$  with  $|S| \geq 4$ .

Suppose that  $\mathcal{L}(R') \neq \mathcal{L}(R)$ . By Lemma 3.14 the graph  $[R, S]$  has at least three components. Now  $[R', S]$  is a subgraph of  $[R, S]$  with the same vertices, so it must also have at least three components.

In a similar way, if  $\mathcal{L}(R') = \mathcal{L}(R)$  and  $S \neq \mathcal{L}(R')$  then  $[R', S]$  has at least three components.

Finally, if  $\mathcal{L}(R') = \mathcal{L}(R)$  and  $S = \mathcal{L}(R')$  then  $[R', S]$  is a subgraph of  $[R, \mathcal{L}(R)]$  with the same vertices and edges missing. Examining the diagram of  $[R, \mathcal{L}(R)]$  in Figure 23 reveals that any such subgraph has at least three components.

By Theorem 3.11,  $R'$  is fully closed. We show that  $R$  is *not* closed. The graphs  $[R \cup \{a_1c_1|b_1\}, \mathcal{L}(R)]$  and  $[R \cup \{b_1c_1|a_1\}, \mathcal{L}(R)]$  are both connected so by Theorem 2.14 (page 37), the sets  $R \cup \{a_1c_1|b_1\}$  and  $R \cup \{b_1c_1|a_1\}$  are incompatible. Hence  $R \vdash a_1b_1|c_1$ , even though  $a_1b_1|c_1 \notin R$ .

It follows that  $R \vdash a_1b_1|c_1$  is a rule that cannot be reduced to repeated application of rules to subsets of  $R$ .  $\square$

An earlier attempt at proving Theorem 3.15 led to a related result, that for any  $k \geq 1$  there exists a set of rooted triples that is incompatible even though every subset of size at most  $k$  is compatible and closed. Of course, by Theorem 3.12 (3), if *every* proper subset of a set of rooted triples is compatible and closed then the entire set is compatible, so we cannot expect a full analogue of Theorem 3.15 to apply here.

Let  $m = 3k + 1$  and put

$$R_3 := \{a_1a_2|b_1, a_2a_3|b_2, \dots, a_{m-1}a_m|b_{m-1}, \\ b_1b_2|c_1, b_2b_3|c_2, \dots, b_{m-1}b_m|c_{m-1}, \\ c_1c_2|a_1, c_2c_3|a_2, \dots, c_{m-1}c_m|a_{m-1}, \\ a_mb_1|b_m, b_mc_1|c_m\}$$

The structure of  $R_3$  is revealed by the associated graph  $[R_3, \mathcal{L}(R_3)]$ , represented in Figure 24.

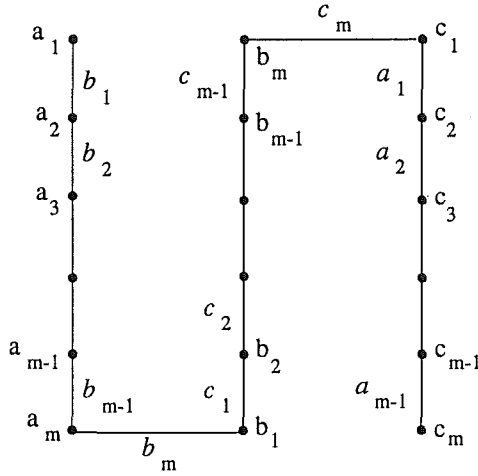


Figure 24 : The graph  $[R_3, \mathcal{L}(R_3)]$ . The edges are labelled in italics. The graph is connected, so  $R_3$  is incompatible by Theorem 2.14.

Using arguments similar to the proofs of Lemma 3.14 and Theorem 3.15, it can be shown that every subset  $R'$  of  $R_3$  with  $|\mathcal{L}(R')| < m$  is both compatible and closed. Hence every subset of  $R_3$  with  $k$  or fewer triples is also compatible and closed, and yet the set  $R_3$  is incompatible, by Theorem 2.14, (and Figure 24). It follows that the set of rooted triple rules of order  $k$  or less is insufficient to determine not only the closure of a set (Theorem 3.15), but also the compatibility of a set. This proves another conjecture of [40].

### 3.5 Closed sets and Quartets

So far as compatibility goes, we have already mentioned how quartets are more difficult to work with than rooted triples. A true pessimist would expect that the attractive properties of rooted triple sets would not hold for quartet sets, and that the unattractive properties of rooted triple sets would hold for quartet sets. And they would be right, though of course who is to say what is attractive? In this case, the properties of rooted triples listed in Theorem 3.12 do not transfer to quartets, at least, two of them do not and the other is uncertain. But the irreducibility result for inference rules does transfer to quartets, though not directly.

#### 3.5.1 Properties (and non-properties) of Quartet Sets

There are exactly two trees compatible with the quartet set  $Q = \{12|36, 23|45, 14|56\}$  as shown in Figure 25. The intersection of the quartet sets of these trees equals  $Q$ , so  $Q$  is closed.

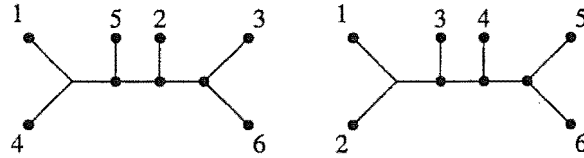


Figure 25 : The two trees in  $\langle 12|36, 23|45, 14|56 \rangle$ .

Note however that neither of these trees are compatible with the quartet  $13|24$ . Hence the set  $Q \cup \{13|24\} = \{12|36, 23|45, 14|56, 13|24\}$  is incompatible, even though  $Q$  is compatible and closed, and there is no quartet with leaf set  $\{1, 2, 3, 4\}$  in  $Q$ . We have shown that property (1) of Theorem 3.12 does not hold for quartets.

The maximal proper subsets of  $Q \cup \{13|24\}$  are  $Q$  and  $\{12|36, 14|56, 13|24\}$  and  $\{12|36, 23|45, 13|24\}$  and  $\{12|36, 23|45, 14|56\}$ . Each of these sets are compatible and closed, even though  $Q \cup \{13|24\}$  is incompatible. We have shown that property (3) of Theorem 3.12 does not hold for quartets.

Property (2) is a bit of an enigma. If we apply inference rules to a set of rooted triples then we obtain a conflict (two triples on the same leaf set) if and only if the

set is incompatible. The question is whether the same applies for quartets. The answer is that we don't know, despite a lengthy and frustrating investigation. We pose two related conjectures, one equivalent to property (2) of Theorem 3.12 and the other a natural extension.

- Conjecture 3.16**
1. *A set  $Q$  is compatible if and only if for all compatible subsets  $Q' \subseteq Q$  and rules  $Q' \vdash ab|cd$ , the set  $Q \cup \{ab|cd\}$  does not contain a quartet conflict.*
  2. *A set  $Q$  is compatible if and only if we cannot obtain a quartet conflict by repeatedly extending  $Q$  using inference rules.*

Note that (1) implies (2) but not vice versa.

### 3.5.2 Links between Quartets and Rooted Triples

Suppose we have a rooted tree  $T$ . Recall from chapter 1, section 1.1.3 that we can convert  $T$  into an unrooted tree  $\text{UNROOT}(T, \rho)$  by attaching the leaf  $\rho$  to the root of  $T$  and unrooting the tree. This operation can be reversed using  $\text{ROOT}(T, \rho)$  which removes the leaf  $\rho$  and makes the adjacent vertex the root of  $T$ .

Using the same principle we can convert a rooted triple  $ab|c$  into a quartet  $ab|c\rho$ . In fact a rooted tree  $T$  extends a rooted triple  $ab|c$  if and only if  $\text{UNROOT}(T, \rho)$  extends  $ab|c\rho$ . This correspondence has a number of useful properties.

**Theorem 3.17** *Let  $R$  be a set of rooted triples. Let  $Q$  be the associated set of quartets:*

$$Q := \{xy|z\rho : xy|z \in R\}$$

*then*

1.  $\langle Q \rangle = \{\text{UNROOT}(T, \rho) : T \in \langle R \rangle\}$
2.  *$Q$  is compatible if and only if  $R$  is compatible.*
3. *If  $Q$  is closed then  $R$  is closed.*
4. *If  $Q \vdash ab|c\rho$  then  $R \vdash ab|c$*

*Proof*

For (1) and (2) we observe that a tree  $T$  extends  $R$  if and only if  $\text{UNROOT}(T, \rho)$  extends  $Q$ .

(3) If  $Q$  is closed then, by Theorem 3.1, there are binary trees  $T_1, \dots, T_k$  such that

$$Q = q(T_1) \cap \dots \cap q(T_k).$$

Put  $T'_i = \text{ROOT}(T_i, \rho)$  for  $i = 1, \dots, k$ . Then each  $T'_i$  extends  $R$  so  $R \subseteq r(T'_1) \cap r(T'_2) \cap \dots \cap r(T'_k)$ . Suppose  $ab|c \in r(T'_1) \cap \dots \cap r(T'_k)$ . Then  $ab|c\rho \in q(T_i)$  for all  $i$ , and so  $ab|c\rho \in Q$ . Hence  $ab|c \in R$ ,  $R = r(T'_1) \cap r(T'_2) \cap \dots \cap r(T'_k)$  and  $R$  is closed by Theorem 3.1.

(4) If  $Q \vdash ab|c\rho$  then  $ab|c\rho \in q(T)$  for all  $T \in \langle Q \rangle$ . Hence by (1),  $ab|c \in r(T')$ ,  $\forall T' \in \langle R \rangle$ , and so  $R \vdash ab|c$ .  $\square$

Thus, if a set of quartets all share one leaf, one can convert the set into a corresponding set of rooted triples and determine in polynomial time whether or not the quartets are compatible.

Note that the converse of Theorem 3.17 (3) is **not** true. For example,  $\{ab|c, ab|d\}$  is a fully closed set of rooted triples, but  $\{ab|c\rho, ab|d\rho\}$  is not a closed set of quartets.

### 3.5.3 Irreducible Quartet Rules

To extend Theorem 3.15 to quartets, we take the set of rooted triples  $R_i$  used to prove the rooted triple case, and convert it into a set of quartets  $Q_i$ , as described in Theorem 3.17. We show that this set is compatible and not closed, even though every proper subset is closed. The fact that the whole set is compatible and not closed follows directly from Theorem 3.17. The fact that each proper subset is closed doesn't. We have already observed that a set of rooted triples can be closed when the corresponding set of quartets is not. To prove our quartet result we take a different route, exploring additional properties of rooted triples.

**Lemma 3.18** *Let  $R$  be any compatible set of rooted triples and suppose that  $R \cup \{ab|c\}$  and  $R \cup \{ab|d\}$  are both compatible. Then  $R \cup \{ab|c, ab|d\}$  is also compatible.*

*Proof*

Let  $S$  be any subset of  $\mathcal{L}(R) \cup \{a, b, c, d\}$ . The sets  $R \cup \{ab|c\}$  and  $R \cup \{ab|d\}$  are

both compatible, so by Theorem 2.14 on page 37 the graph  $[R \cup \{ab|c\}, S]$  and the graph  $[R \cup \{ab|d\}, S]$  are both disconnected.

If  $a \notin S$  or  $b \notin S$  then the graph  $[R \cup \{ab|c, ab|d\}, S]$  is the same as the graph  $[R, S]$ , so is disconnected (Theorem 2.14).

If  $c \notin S$  then the graph  $[R \cup \{ab|c, ab|d\}, S]$  is the same as the graph  $[R \cup \{ab|d\}, S]$ , so is disconnected. By symmetry, if  $d \notin S$  then  $[R \cup \{ab|c, ab|d\}, S]$  is disconnected. Finally, if  $a, b, c$  and  $d$  are all in  $S$ , then the graph  $[R \cup \{ab|c, ab|d\}, S]$  is the same as the graph  $[R \cup \{ab|c\}, S]$  with an extra label  $d$  on the edge  $(a, b)$ , so the graph is still disconnected. In any of these five cases, the graph  $[R \cup \{ab|c, ab|d\}, S]$  is disconnected. Hence  $R \cup \{ab|c, ab|d\}$  is compatible, by Theorem 2.14.  $\square$

**Theorem 3.19** *Given any  $n \geq 1$  there is a compatible set of  $n$  quartets that is not closed even though every proper subset is closed. Thus there is a quartet rule of order  $n$  that cannot be derived by repeated application of rules of order less than  $n$ .*

*Proof*

When  $n = 1, 2$  the proof is trivial. If  $n = 3$  then a suitable example is  $\{ab|cd, ab|ef, ce|df\}$  [40]. When  $n > 3$  define the set  $R(n)$  as follows:

If  $n \equiv 0 \pmod{3}$  then put  $m = n/3$  and  $R(n) = R_0$ .

If  $n \equiv 1 \pmod{3}$  then put  $m = (n - 1)/3$  and  $R(n) = R_1$ .

If  $n \equiv 2 \pmod{3}$  then put  $m = (n + 1)/3$  and  $R(n) = R_2$ .

The sets  $R_0, R_1$  and  $R_2$  are given by equations 3, 4 and 5 on page 66. Note that this definition of  $R(n)$  is identical to the definition of  $R(n)$  on page 68.

Construct the set of quartets

$$Q := \{xy|z\rho : xy|z \in R\}.$$

For example, when  $n \equiv 1 \pmod{3}$ :

$$\begin{aligned} Q := & \{a_1a_2|b_2\rho, a_2a_3|b_3\rho, \dots, a_ma_{m+1}|b_{m+1}\rho, \\ & b_1b_2|c_1\rho, b_2b_3|c_2\rho, \dots, b_mb_{m+1}|c_m\rho, \\ & c_1c_2|a_1\rho, c_2c_3|a_2\rho, \dots, c_mc_{m+1}|a_m\rho, \\ & a_{m+1}b_1|c_{m+1}\rho\} \end{aligned}$$

By Theorem 3.17,  $Q$  is not closed. We claim that every proper subset of  $Q$  is closed.



Let  $Q'$  be a proper subset of  $Q$  and let  $R'$  be the corresponding subset of  $R$ . Consider any four leaves  $a, b, c$  and  $d$  in  $\mathcal{L}(R') = \mathcal{L}(Q') - \{\rho\}$ . First of all we show that there is no quartet with leaves  $\{a, b, c, d\}$  in the closure of  $Q'$ .

No two triples in  $R$  have more than one leaf in common so there is at most one triple in  $R$ , and therefore in  $R'$ , with all its leaves in  $\{a, b, c, d\}$ . If there is such a triple in  $R'$  we assume, without loss of generality, that this is the triple  $ab|c$ . Hence there are no triples in  $R$  (and hence in  $R'$ ) with leaves  $\{a, c, d\}$ ,  $\{b, c, d\}$  or  $\{a, b, d\}$ . Of course this also applies if there is no triple in  $R'$  with leaves in  $\{a, b, c, d\}$ .

By Theorem 3.12 (1) we have

$$(i) \ R' \cup \{cd|a\}, R' \cup \{cd|b\} \text{ are both compatible}$$

and

$$(ii) \ R' \cup \{ad|c\}, R' \cup \{ad|b\} \text{ are both compatible}$$

Applying Lemma 3.18 to (i), the set  $R' \cup \{cd|a, cd|b\}$  is compatible, so by Theorem 3.17 (2), the set  $Q' \cup \{cd|a\rho, cd|b\rho\}$  is compatible. But  $\{cd|a\rho, cd|b\rho\} \vdash cd|ab$ , so  $Q' \cup \{cd|ab\}$  is compatible.

By Lemma 3.18 and (ii), the set  $R' \cup \{ad|c, ad|b\}$  is compatible, so by Theorem 3.17 the set  $Q' \cup \{ad|c\rho, ad|b\rho\}$  is compatible. But  $\{ad|c\rho, ad|b\rho\} \vdash ad|bc$ , so  $Q' \cup \{ad|cb\}$  is compatible. Since  $Q' \cup \{cd|ab\}$  is also compatible, there is no quartet with leaves  $\{a, b, c, d\}$  in the closure of  $Q'$ .

Thus, to prove that  $Q'$  is closed we only need to show now that there are no quartets of the form  $ab|c\rho$  in the closure of  $Q'$  that are not already contained in  $Q'$ . If  $Q' \vdash ab|c\rho$  then by Theorem 3.17 (4),  $R' \vdash ab|c$ . As  $R'$  is closed, this implies that  $ab|c \in R'$ , and so  $ab|c\rho \in Q'$ .

Hence  $Q'$  is closed. We show that  $Q$  itself is not closed. Recall from the proof of Theorem 3.15 that  $R \vdash a_1b_1|c_1$ . By Theorem 3.17 (4),  $Q \vdash a_1b_1|c_1\rho$ . It follows that  $Q \vdash a_1b_1|c_1\rho$  is a rule of order  $n$  that cannot be derived through repeated application of rules with order less than  $n$ .  $\square$



## Chapter 4

# Hunting for Trees - Character Data

### 4.1 Introduction

There is no shortage of tree-building methods and each has its advantages and disadvantages, advocates and critics. The methods fall roughly into two categories: those applied to discrete character data and those applied to distance data [84]. In this chapter and the following we look at examples of both types. We describe new and efficient algorithms for existing methods, as well as proposing variants and new techniques.

First we consider the methods for character data, in particular, binary (two-state) character data. Binary characters are a common starting point for phylogenetic analysis. They can be obtained, for example, from purine/pyrimidine genetic data or morphological distinctions like vertebrate/invertebrate. However the characters need not come from molecular or phylogenetic data at all. The characters can be derived from distances (see Chapter 5) or extracted from profiles of trees as part of consensus methods (see Chapter 6).

A set of binary characters is compatible if the corresponding set of splits is compatible. If the characters are compatible then we can append the trivial splits and represent the character set exactly using a single tree. More often, the characters are not compatible and we have to choose a tree that is 'best' according to some criterion. This might involve some measure of how well each character fits a tree,

as in parsimony, or a measure of how many of the input characters fit into a tree, as in the various compatibility methods.

Compatibility methods are employed throughout systematics. They appear under a number of guises: in cladistics [115], numerical taxonomy [73], evolutionary systematics [44], and molecular biology [69]. One common method is to infer phylogenies by finding those phylogenies compatible with the largest number of characters, or, if the characters are weighted, finding the phylogenies compatible with a subset of characters of largest combined weight. Unfortunately this general problem is NP-hard, even for binary characters [39]. Researchers have used the method only on small data sets, or resigned themselves to heuristic algorithms that do not guarantee the best solution. A number of related algorithms and programs are available (see [83]). Our approach is to slightly modify the problem, giving a compatibility method that is useful, non-trivial, and has a polynomial time algorithm.

We first describe the algorithm not for binary characters and unrooted trees but for clusters and rooted trees. A simple transformation then gives us an algorithm for the unrooted case. We then discuss a variety of extensions to the algorithm, together with applications and examples. We call this collection of algorithms the *Hunting for Trees* algorithms.

## 4.2 A special case: sets of clusters

It is time for a formal declaration of the problem solved. Recall from Chapter 1 that a rooted  $d$ -tree is a rooted tree in which no vertex has more than  $d$  children.

### HUNTING FOR TREES IN CLUSTERS

INSTANCE: Collection  $C$  of subsets of a leaf set  $X$  such that  $X \in C$ . Number  $d \geq 2$ .

PROBLEM: Determine if there is a subcollection  $C' \subseteq C$  such that  $C' = \sigma(T)$  for some rooted phylogenetic  $d$ -tree  $T$  with leaf set  $X$ .

Our solution is based on the following:

**Observation** *A given collection  $C$  of clusters contains a subcollection corresponding*

to a  $d$ -tree with leaf set  $X$  if and only if  $X \in C$  and  $X$  equals the union of at most  $d$  disjoint clusters in  $C$ , which in turn equal the union of at most  $d$  disjoint clusters in  $C$  and so on, right down to the level of singletons.

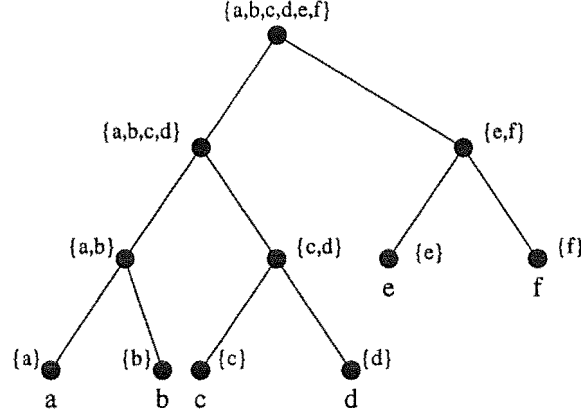


Figure 26 : A rooted binary phylogenetic tree with leaf set  $\{a, b, c, d, e, f\}$ .

Consider the tree in Figure 26. The internal vertices are labelled with their respective clusters. The leaf set  $\{a, b, c, d, e, f\}$  is the union of two disjoint clusters,  $\{a, b, c, d\}$  and  $\{e, f\}$ , which in turn equal the union of two disjoint clusters and so on, until we reach the leaves of the tree.

#### 4.2.1 Principal algorithm

Let  $k$  be the number of clusters in  $C$  and  $n$  be the number of taxa in  $X$ . We assume that  $C$  contains all the single element clusters. In practice these can simply be appended to the input data.

The procedure `CREATETABLE` (Algorithm 3) creates a table  $D$ , called the **decomposition table**, which is later used to count and optimise the  $d$ -trees with clusters in  $C$  and leaf set  $X$ .

Steps 1, 2 and 3 can be completed in  $O(nk)$  time using a radix sort [60, 5]. For arbitrary  $\{p_1, p_2, \dots, p_j\}$ , ( $j \leq d$ ), determining whether  $C_{p_1}, \dots, C_{p_j}$  are pairwise disjoint, and calculating  $C_{p_1} \cup \dots \cup C_{p_j}$  takes at most  $O(nd)$  time. Since  $C$  is well-ordered it takes just  $O(\log K)$  time to determine whether  $C_{p_1} \cup \dots \cup C_{p_j}$  is in  $C$ , which

Procedure CREATETABLE( $C, d$ )

1. Represent each cluster in  $C$  by an  $n$ -digit binary number, with a 1 for the  $i$ th digit if the  $i$ th taxon is in the cluster, and a 0 otherwise (as in a characteristic vector).
  2. Sort the clusters into ascending order of their associated binary numbers.
  3. Remove duplicates and label remaining clusters  $\{C_1, \dots, C_K\}$ .
  4. Create a table  $D$  consisting of  $K$  empty lists labelled  $D[1], \dots, D[K]$ .
  5. FOR all subsets  $\{p_1, p_2, \dots, p_j\}$  of  $\{1, \dots, K\}$  with at least two and at most  $d$  elements DO
  6.   IF  $C_{p_1}, \dots, C_{p_j}$  are pairwise disjoint AND  $C_{p_1} \cup \dots \cup C_{p_j} = C_q$  for some  $q \in \{1, \dots, K\}$  THEN append  $\{p_1, p_2, \dots, p_j\}$  to the list  $D[q]$
  7. END(FOR)
- END.

Algorithm 3: CREATETABLE

is just  $O(n)$  time since  $K \leq 2^n$ . Hence steps 5 to 7 can be completed in  $O(ndK^d)$  time, where  $K$  is the number of *distinct* clusters in  $C$ ; the procedure CREATETABLE takes at most  $O(nk + ndK^d)$  time. Note that  $K$  is often significantly smaller than  $k$ .

To illustrate the algorithm we consider a simple example. Put

$$C = \{\{1\}, \{2\}, \{3\}, \{1, 2, 3\}, \{4\}, \{5\}, \{4, 5\}, \{1, 2, 3, 4, 5\}\},$$

a set of clusters on the leaf set  $\{1, 2, 3, 4, 5\}$ . When  $d = 3$ , Algorithm 3 generates the following decomposition table.

$i$	$C_i$	$D[i]$
1	{1}	$\emptyset$
2	{2}	$\emptyset$
3	{3}	$\emptyset$
4	{1, 2, 3}	{{1, 2, 3}}
5	{4}	$\emptyset$
6	{5}	$\emptyset$
7	{4, 5}	{{5, 6}}
8	{1, 2, 3, 4, 5}	{{4, 7}, {4, 5, 6}}

### 4.2.2 Counting the Trees

For each  $i = 1, \dots, K$  Algorithm 4 below calculates the number  $m[i]$  of  $d$ -trees with leaf set  $C_i$  and clusters contained in  $C$ .

Procedure COUNTTREES( $D$ )

1. FOR  $i$  from 1 to  $K$  DO
  2.   IF  $C_i$  is a singleton THEN  $m[i] \leftarrow 1$
  3.   ELSE IF  $D[i]$  is empty THEN  $m[i] \leftarrow 0$
  4.   ELSE
 
$$m[i] \leftarrow \sum_{\{p_1, p_2, \dots, p_j\} \in D[i]} m[p_1] \times m[p_2] \times \dots \times m[p_j].$$
  5.   Remove from  $D[i]$  all tuples  $\{p_1, \dots, p_j\}$  such that one of  $m[p_1], \dots, m[p_j]$  equals 0.
  6. END(FOR)
- END.

Algorithm 4 : COUNTTREES

**Theorem 4.1** *After completion of Algorithm 4,  $m[i]$  equals the number of distinct rooted  $d$ -trees with clusters in  $C$  and leaf set  $C_i$ .*

*Proof*

We proceed by induction. The theorem is clearly true when  $i = 1$ . Suppose that it is true when  $i = 1, \dots, j-1$ . Let  $\mathcal{T}_j$  be the set of rooted  $d$ -trees with leaf set  $C_j$  and clusters contained in  $C$ . We will show that  $m[j] = |\mathcal{T}_j|$ .

Let  $T \in \mathcal{T}_j$ , and let  $q$  be the number of maximal subtrees in  $T$ . The leaf sets of these subtrees are equal to a set of  $q$  pairwise disjoint clusters  $\{C_{p_1}, \dots, C_{p_q}\}$  in  $\sigma(T)$ , and therefore in  $C$ , such that  $C_{p_1} \cup \dots \cup C_{p_q} = C_j$ . The tuple  $\{p_1, \dots, p_q\}$  is therefore contained in  $D[j]$ . It follows that we can partition  $\mathcal{T}_j$  so that each block in the partition corresponds to a different tuple in  $D[j]$ . Hence

$$|\mathcal{T}_j| = \sum_{\{p_1, \dots, p_q\} \in D[j]} (\# \text{ trees in } \mathcal{T}_j \text{ containing the clusters } C_{p_1}, \dots, C_{p_q}).$$

Fix any such tuple  $\{p_1, \dots, p_q\} \in D[j]$ . The number of trees in  $\mathcal{T}_j$  that contain clusters  $C_{p_1}, \dots, C_{p_q}$  equals the product  $|\mathcal{T}_{p_1}| \times \dots \times |\mathcal{T}_{p_q}|$ . By the induction hypothesis this equals  $m[p_1] \times \dots \times m[p_q]$ . Therefore,

$$\begin{aligned} |\mathcal{T}_j| &= \sum_{\{p_1, \dots, p_q\} \in D[j]} m[p_1] \times m[p_2] \times \dots \times m[p_q] \\ &= m[j] \end{aligned}$$

as required.  $\square$

Each tuple  $\{p_1, p_2, \dots, p_j\}$  in  $D[i]$  satisfies  $C_{p_1} \cup \dots \cup C_{p_j} = C_i$ , so there are at most  $O(K^{d-1})$  such tuples in each list  $D[i]$ . Hence calculating  $m[i]$  for all  $i$  takes  $O(K^d)$  time.

In the previous section (page 80) we constructed the decomposition table for

$$C = \{\{1\}, \{2\}, \{3\}, \{1, 2, 3\}, \{4\}, \{5\}, \{4, 5\}, \{1, 2, 3, 4, 5\}\},$$

with  $d = 3$ . If we now apply Algorithm 4 to this decomposition table we can calculate the entries of  $m[i]$ .



$i$	$C_i$	$D[i]$	$m[i]$
1	{1}	$\emptyset$	1
2	{2}	$\emptyset$	1
3	{3}	$\emptyset$	1
4	{1, 2, 3}	{{1, 2, 3}}	1
5	{4}	$\emptyset$	1
6	{5}	$\emptyset$	1
7	{4, 5}	{{5, 6}}	1
8	{1, 2, 3, 4, 5}	{{4, 7}, {4, 5, 6}}	2

Thus there are exactly two 3-trees with clusters in  $C$ .

### 4.2.3 Tree extraction

It is now a simple matter to list the subcollections  $C'$  that correspond to  $d$ -trees. Given a collection of clusters  $C$ , we say that a particular cluster  $C_i \in C$  is **unresolved** if there is no cluster  $C_j \in C$  such that  $C_j \subset C_i$ ,  $C_j \neq C_i$ . The following non-deterministic recursive procedure, GETSUBCOLLECTIONS, returns an arbitrary one of these subcollections. At each level it chooses from the list an unresolved cluster  $C_i$  and adds to the list pairwise disjoint clusters  $C_{p_1}, \dots, C_{p_j}$  such that  $C_{p_1} \cup \dots \cup C_{p_j} = C_i$ .

The first parameter is the list being built up, initially just  $\{C_K\}$ , and the second parameter is the table  $D$  constructed by the procedure CREATETABLE and pruned by the procedure COUNTTREES.

Procedure GETSUBCOLLECTIONS( $C', D$ )

1. IF the only unresolved clusters in  $C'$  are singletons THEN
2.     Output the subcollection  $C'$
3. ELSE
4.     Choose any unresolved cluster  $C_i \in C'$  that is not a singleton
5.     Choose  $\{p_1, \dots, p_j\}$  in  $D[i]$
6.     GETSUBCOLLECTIONS( $C' \cup \{C_{p_1}, \dots, C_{p_j}\}, D$ )
7. END(IF-ELSE)

END.

Algorithm 5 : GETSUBCOLLECTIONS

If  $C'$  is the set of clusters of some tree  $T$ , then the graph of  $T$  can be easily retrieved directly from  $C'$  in  $O(n|C'|) = O(n^2)$  time [60]. Since it takes only  $O(|C'|)$  time to construct  $C'$ , extracting a single  $d$ -tree takes only  $O(n^2)$  time.

It is not always practical to list all of the subcollections of  $C$  that correspond to binary trees, as the following theorem illustrates.

**Theorem 4.2** *Given a set  $X$  with  $n$  taxa, there exists a collection  $C$  with  $(n^2 + n)/2$  clusters such that the number of subcollections  $C' \subseteq C$  corresponding to rooted binary trees ( $d = 2$ ) with leaf set  $X$  exceeds  $2^{n-2}$ .*

*Proof*

Given  $n$ , let  $X = \{1, 2, \dots, n\}$  and construct

$$C(n) = \{C_{a,b} : 1 \leq a \leq b \leq n, \text{ where } a, b \in X\}$$

where

$$C_{a,b} = \{x \in X : a \leq x \leq b\}.$$

Then  $|C(n)| = (n^2 + n)/2$ . Let  $\mathcal{T}(n)$  be the set of binary trees with leaf sets  $X$  and clusters in  $C(n)$ . We claim that  $|\mathcal{T}(n)| > 2^{n-2}$ .

The result is easily verified for  $n = 1, 2, 3$ . For the induction step suppose that  $|\mathcal{T}(m)| > 2^{m-2}$ . We show that  $|\mathcal{T}(m+1)| > 2^{(m+1)-2}$ . Given any tree  $T$  in  $\mathcal{T}(m)$ , we construct two new trees in  $\mathcal{T}(m+1)$ .

1. Create a new root with two descendants. Let one descendant be the root of  $T$ , and the other descendant be the leaf  $(m+1)$ .
2. Replace leaf  $m$  of  $T$  with a new vertex. Let the leaves  $m$  and  $m+1$  be the two descendants of this new vertex.

Both of these trees are binary and have clusters in  $C(m+1)$ . Hence for every tree in  $\mathcal{T}(m)$  there are at least two trees in  $\mathcal{T}(m+1)$ , all of which are distinct. By the induction hypothesis

$$|\mathcal{T}(m+1)| \geq 2 \times |\mathcal{T}(m)| > 2 \times 2^{m-2} = 2^{(m+1)-2}$$

as required.  $\square$

#### 4.2.4 Optimisation

We can use the dynamical programming technique for counting trees to solve the following, seemingly more difficult, problem.

##### HUNTING FOR TREES IN WEIGHTED CLUSTERS

INSTANCE: Collection  $C$  of subsets of a finite set  $X$ . Weighting function  $w : C \rightarrow \mathfrak{R}$ . Number  $d \geq 2$ .

PROBLEM: Find a subcollection  $C' \subseteq C$  that maximises  $\sum_{C_i \in C'} w(C_i)$  such that  $C' = \sigma(T)$  for some rooted  $d$ -tree  $T$ .

The following procedure, MAXWEIGHTTREE, takes three parameters: the first parameter is the collection  $C$ ; the second parameter is the table  $D$  constructed by the procedure CREATETABLE and pruned in COUNTTREES; the third is the table  $m$  constructed with the procedure COUNTTREES.

Procedure MAXWEIGHTTREE( $C, D, m$ )

1. FOR  $i$  from 1 to  $K$  DO
2.   IF  $m[i] \neq 0$  THEN
3.     IF  $C_i$  is a singleton THEN
4.        $M[i] \leftarrow w(C_i)$
5.        $D^*[i] \leftarrow \emptyset$
6.        $m^*[i] \leftarrow 1$
7.     ELSE
8.       Choose  $\{p_1, \dots, p_j\}$  in  $D[i]$  that maximises  $M[p_1] + \dots + M[p_j]$
9.        $M[i] \leftarrow M[p_1] + \dots + M[p_j] + w(C_i)$
10.        $D^*[i] \leftarrow \{\{q_1, \dots, q_r\} : M[q_1] + \dots + M[q_r] + w(C_i) = M[i]\}$
11.        $m^*[i] \leftarrow \sum_{\{q_1, \dots, q_r\} \in D^*[i]} m^*[q_1] \times \dots \times m^*[q_r]$
12.     END(IF-ELSE)
13.   END(IF)
14. END(FOR)

END.

Algorithm 6 : MAXWEIGHTTREE

The procedure returns three tables,  $M$ ,  $D^*$  and  $m^*$ . For each  $i$ ,  $M[i]$  is the weight of the maximum weight subcollection that corresponds to a  $d$ -tree with leaf set  $C_i$ . The table  $D^*$  is a reduced version of  $D$  used to list the maximum weight subcollections. The table  $m^*$  is used to count the number of maximum weight subcollections.

The procedure has complexity  $O(K^d)$ , where  $K$  is the number of distinct clusters in  $C$ . The number of optimal subcollections equals  $m^*[K]$ . To output an arbitrary optimal subcollection, call  $\text{GETSUBCOLLECTIONS}(\{C_K\}, D^*)$ . There are sometimes, however, an exponentially large number of optimal subcollections. Correctness is given by the following theorem.

**Theorem 4.3** *After execution of the procedure MAXWEIGHTTREE, if  $m[i] \neq 0$  then  $M[i]$  equals the weight of the maximum weight  $d$ -tree with leaf set  $C_i$  and clusters in  $C$ .*

*Proof*

Again we let  $\mathcal{T}_i$  denote the set of  $d$ -trees with leaf set  $C_i$  and clusters in  $C$ . Let  $w(T)$  denote the sum of the weights of the clusters of a phylogenetic tree  $T$ . We need to prove two things:

- (i)  $w(T) \leq M[i]$  for all  $T \in \mathcal{T}_i$ .
- (ii) There is  $T \in \mathcal{T}_i$  such that  $w(T) = M[i]$ .

We proceed by induction. If  $i = 1$  then  $C_i$  is a leaf, so (i) and (ii) hold. Suppose that (i) and (ii) are true for  $i = 1, \dots, j - 1$ .

Let  $T \in \mathcal{T}_j$ . There is  $\{p_1, \dots, p_q\} \in D[j]$  such that  $C_{p_1}, \dots, C_{p_q}$  are maximal clusters of  $T$ . Now  $C_{p_1} \cup \dots \cup C_{p_q} = C_j$ , so the  $q$  clusters correspond to the  $q$  subtrees branching off the root of  $T$ . Applying the induction hypothesis to the subtrees of  $T$ , we obtain

$$w(T) \leq M[p_1] + \dots + M[p_q] + w(C_j) \leq M[j],$$

proving (i).

For (ii), let  $\{p_1, \dots, p_q\}$  be a tuple in  $D[j]$  such that

$$M[p_1] + \dots + M[p_q] + w(C_j) = M[j].$$

By the induction hypothesis for each  $s : 1 \leq s \leq q$  there is a tree  $T_{p_s}$  with leaf set  $C_{p_s}$  and clusters in  $C$  such that  $w(T_{p_s}) = M[p_s]$ . Construct a new tree  $T$  by connecting a new root  $r$  to the roots of  $T_{p_1}, \dots, T_{p_q}$ . Then  $T \in \mathcal{T}_j$  and  $w(T) = M[j]$ .  $\square$

#### 4.2.5 Selecting a degree bound

Given a set of clusters  $C$  we wish to determine the smallest value of  $d$  such that  $C(T) \subseteq C$  for some  $d$ -tree  $T$ . Once this number is obtained we can guarantee that the above algorithms will return a tree. Unfortunately the following theorem suggests that there is no efficient way to determine this minimal  $d$  value.

**Theorem 4.4** *Let  $C$  be a set of clusters. The problem of finding the smallest  $d$  such that  $C$  contains the clusters of a  $d$ -tree is NP-hard.*

*Proof*

We transform EXACT COVER BY 3-SETS (X3C) [55] to the minimum  $d$  problem. Let  $X$ ,  $|X| = 3q$ , be a set and  $C$  be a collection of 3-element subsets of  $X$  making up an arbitrary instance of X3C. Take  $X$  as a set of leaves and construct  $C' = C \cup \{\{x\} : x \in X\} \cup \{X\}$ . If  $X$  has an exact cover by a subcollection of  $C$  then the same subsets, together with  $X$  and the singleton clusters, determine a  $q$ -tree with clusters in  $C'$ . Conversely if  $C''$  is a subcollection of  $C'$  that determines a  $q$ -tree then the non-trivial clusters in  $C''$  give an exact covering of  $X$ . Therefore there is an exact covering of  $X$  by a subcollection of  $C$  if and only if the minimum  $d$  such that  $C'$  contains a  $d$ -tree equals  $q$ . It follows that the minimum  $d$  problem is NP-Hard.  $\square$

Because of Theorem 4.4 we should expect any solution to the minimum degree bound problem to have exponentially increasing complexity. The algorithm MINIMUM $d$  below has running time  $O(nk + n\delta K^\delta)$  where  $n$  is the number of leaves,  $k$  is the number of clusters,  $K$  is the number of distinct clusters, and  $\delta$  is the minimum

Procedure MINIMUMd( $C$ )

1. Sort  $C$  and remove duplicates. Let  $K$  be the number of distinct clusters.
2. FOR  $d$  from 2 to  $n$  DO
3.     Construct  $D$  using CREATETABLE( $C, d$ )
4.     Construct  $m$  using COUNTTREES( $C$ )
5.     IF  $m[K] > 0$  THEN
6.          $\delta \leftarrow d$
7.     RETURN  $\delta$ .
8.     END(IF)
9. END(FOR)
- END.

Algorithm 7 : MINIMUMD

value of  $d$  such that  $C$  contains the splits of a  $d$ -tree. It therefore takes polynomial time with respect to the number of leaves and the number of characters, but takes exponentially increasing time with respect to the minimum value of  $d$  recovered.

#### 4.2.6 Hunting through all possible clusters

There are some situations, usually with small or medium sized leaf sets, where we would want to hunt for trees in the complete set of all possible clusters. If  $|L| = n$  then the number of clusters in this set equals  $2^n$  so we would expect to take time  $O(n2^n + n(2^n)^2)$  time to search through all binary trees on that leaf set, using the above algorithms.

However we can achieve a better bound.

**Theorem 4.5** *Let  $w$  be a weighting on the clusters of  $L$ . A binary tree having clusters of maximum summed weight can be retrieved in  $O(3^n)$  time.*

*Proof*

Process the clusters in order of increasing size. For each cluster  $C_i$  we check the

$2^{|C_i|-1}$  different ways that  $C_i$  can be partitioned into two non-empty parts, summing the maximum weight in each case. Hence determining the maximum weight tree with leaf set equal to a particular cluster  $C_i$  takes  $O(2^{|C_i|-1})$  time, and calculating the maximum weight tree with leaves in  $L$  takes

$$O\left(\sum_{k=1}^n \binom{n}{k} 2^{k-1}\right) \text{ time.}$$

This is  $O(3^n)$  time since

$$\frac{1}{2}3^n = \frac{1}{2}(1+2)^n = \frac{1}{2} \sum_{k=0}^n \binom{n}{k} 2^k = \sum_{k=0}^n \binom{n}{k} 2^{k-1}. \quad \square$$

A consequence of Theorem 4.5 is that if  $d$  is big and  $n$  is small then it is sometimes more efficient to check all possible clusters. The result also gives a practical solution to a problem in spectral analysis. The final step in spectral analysis (see Hendy and Penny [65] for details) is to construct a maximum weight tree from a set of weighted splits, which as we show shortly is the same as constructing a maximum weight rooted tree from a set of weighted clusters. In many cases the data will be such that a branch and bound algorithm can quickly find a global optimum. However if the data is messy and there are a lot of closely weighted splits then the *Hunting for Trees* methods may be preferable. We have shown that they guarantee a global optimum in at most  $O(3^n)$  time.

### 4.3 Sets of binary characters

We now consider sets of binary characters. Since a binary character is just a split in disguise we will treat the two as synonymous. In terms of splits our two main problems are:

#### HUNTING FOR TREES IN SPLITS

INSTANCE: Collection  $S$  of splits on leaf set  $L$ . Number  $d \geq 3$ .

PROBLEM: Determine if there is a subcollection  $S' \subseteq S$  such that  $S' = \beta(T)$  for some unrooted phylogenetic  $d$ -tree  $T$ .

#### HUNTING FOR TREES IN WEIGHTED SPLITS

INSTANCE: Collection  $S$  of splits on leaf set  $L$ . Weighting function  $w : S \rightarrow \mathbb{R}$ .

Number  $d \geq 3$ .

PROBLEM: Find a subcollection  $S' \subseteq S$  that maximises  $\sum_{S_i \in S'} w(S_i)$  such that  $S' = \beta(T)$  for some unrooted phylogenetic  $d$ -tree  $T$ .

Recall that an unrooted  $d$  tree is an unrooted tree with maximum vertex degree  $d$  (page 6).

In Chapter 1, section 1.1.3 we saw that an unrooted tree could be transformed into an equivalent rooted tree and back using the operations  $\text{ROOT}(T, x)$  and  $\text{UNROOT}(T, x)$ . Later, in Chapter 3 section 3.5.2 we showed that the same operations could be extended to quartets and rooted triples. Now we show that they also apply to splits and clusters and use them to solve the two 'HUNTING FOR TREES IN SPLITS' problems above.

Given a split  $A|B$  of leaf set  $L$  and a leaf  $\rho \in L$ , define

$$\text{ROOT}(A|B, \rho) = \begin{cases} A & \text{if } \rho \in B \\ B & \text{if } \rho \in A \end{cases}$$

Likewise, if  $A$  is a cluster of  $L$  and  $\rho \notin L$  define

$$\text{UNROOT}(A, \rho) = A|(L - A) \cup \{\rho\}.$$

The operations are extended to sets of splits or clusters in the obvious way. We immediately get

**Theorem 4.6** *Let  $T$  be a rooted tree, choose  $\rho \notin L$  and put  $T' = \text{UNROOT}(T, \rho)$ . Hence  $T = \text{ROOT}(T', \rho)$ . Then*

$$\begin{aligned} \beta(T') &= \text{UNROOT}(\sigma(T), \rho) \\ \text{and } \sigma(T) &= \text{ROOT}(\beta(T'), \rho). \end{aligned}$$

If  $\rho \in L$  then a set of splits  $S$  on  $L$  equals the set of splits for some  $d$ -tree if and only if  $\text{ROOT}(S, \rho)$  equals the set of clusters of some  $(d - 1)$ -tree.

Given a set  $S$  of  $k$  splits on a set  $L$ , choose an arbitrary  $\rho \in L$  and calculate  $\text{ROOT}(S, \rho)$ . This can be done in  $O(nk)$  time. Apply the above cluster algorithms to  $\text{ROOT}(S, \rho)$  to count the total number of subcollections corresponding to  $(d - 1)$ -trees. A particular subcollection  $C' \subset \text{ROOT}(S, \rho)$ , such as one returned by the



algorithm GETSUBCOLLECTIONS, can be transformed into the corresponding set of splits using UNROOT( $C', \rho$ ).

If the original set  $S$  of splits was weighted then weight the elements of  $A \in \text{ROOT}(S, \rho)$  by  $w(A) = w(\text{UNROOT}(A, \rho))$ . If  $C'$  is an optimal subcollection of  $\text{ROOT}(S, \rho)$  then  $\text{UNROOT}(C', \rho)$  will be an optimal subcollection of  $S$ .

## 4.4 Decomposition table bag of tricks

The guts of the above algorithms is the decomposition table. We have seen that the decomposition table can be used to store, in polynomial time, all of the  $d$ -trees with splits in a particular set. Once stored we can count the trees and optimise over them.

The decomposition table data structure provides a compact representation of large sets of trees. If a set  $\mathcal{T}$  of trees contains only binary trees, or more generally, only  $d$ -trees, then the size of the representative decomposition table is  $O(nK^2)$  or  $O(K^d)$  respectively, where  $K = |\cup_{T \in \mathcal{T}} \beta(T)|$ . The size of  $\mathcal{T}$  itself could be exponentially large.

Which sets of trees can be stored in this way? One example is the set of binary trees that extend a given set of quartets, since a binary tree  $T$  is in  $\langle Q \rangle$  if and only if  $\beta(T) \subseteq CS(Q)$ . In general, a set  $\mathcal{T}$  of rooted trees on leaf set  $L$  can be stored in a decomposition table if and only if

If  $T_1, T_2 \in \mathcal{T}$  and  $X \in \sigma(T_1) \cap \sigma(T_2)$  then the tree constructed by replacing the subtree  $T_{1|X}$  of  $T_1$  with the subtree  $T_{2|X}$  is also in  $\mathcal{T}$ .

An equivalent condition applies to unrooted trees.

Here we describe several tricks we can perform on sets of trees stored in a decomposition table, whether this set originates from a *Hunting for Trees* problem or elsewhere. We can find the intersection of two decomposition tables, count and extract trees in a decomposition table that contain specified splits, find the rooted triples or quartets common to the trees, and even generate a number of different consensus trees. Later, in Chapter 5, we show how to select the tree with the smallest minimum evolution score. All of the methods run in polynomial time. First, some notation. Given a leaf set  $L$  and a decomposition table  $D$  let  $\mathcal{T}(D)$  be the set

of trees with leaf sets  $L$  that can be extracted from  $D$ . If  $C$  is a collection of clusters of  $L$  and  $d \geq 2$  then let  $\mathcal{T}(C, d)$  denote the set of rooted  $d$ -trees with clusters in  $C$  and leaf set  $L$ . If  $S$  is a set of splits of  $L$  and  $d \geq 3$  then let  $\mathcal{T}(S, d)$  denote the set of unrooted  $d$ -trees with splits in  $S$ .

#### 4.4.1 Intersection of Decomposition tables

If  $D_1$  and  $D_2$  are two decomposition tables with the same set of leaves then is there some way that we can calculate  $\mathcal{T}(D_1) \cap \mathcal{T}(D_2)$  without calculating all of  $\mathcal{T}(D_1)$  and  $\mathcal{T}(D_2)$ ? Yes there is, using Algorithm 8.

Procedure INTERSECTD( $D_1, D_2$ )

1. Delete the rows of  $D_1$  and  $D_2$  that correspond to clusters not present in the other table. Delete the tuples containing references to these rows.
  2. Renumber the rows of  $D_2$  and the respective entries in the tuples of  $D_2$  so that the row  $i$  of  $D_2$  corresponds to the same cluster as row  $i$  of  $D_1$ .
  3. FOR every row  $i$  of  $D_1$  DO
  4.      $D[i] \leftarrow D_1[i] \cap D_2[i]$ .
  5. END(FOR)
  6. return  $D$ .
- end.

Algorithm 8 : INTERSECTD

**Theorem 4.7** *If  $D$  is the decomposition table returned by INTERSECTD( $D_1, D_2$ ) then  $\mathcal{T}(D) = \mathcal{T}(D_1) \cap \mathcal{T}(D_2)$ .*

*Proof*

If  $T \in \mathcal{T}(D_1) \cap \mathcal{T}(D_2)$  then the clusters of  $T$  each correspond to a row in  $D_1$  and a row in  $D_2$ . These rows are not deleted in step 1. As well, the tuples in  $D_1$  that are used when extracting  $T$  must also be tuples in  $D_2$  so they are not deleted in step 3. Hence  $\mathcal{T}(D_1) \cap \mathcal{T}(D_2) \subseteq \mathcal{T}(D)$ .

Conversely, if  $T \notin \mathcal{T}(D_2)$  then either there is a cluster of  $T$  not corresponding to a row of  $D_2$ , in which case  $T$  gets excluded in step 1, or there is a tuple of clusters in  $T$  not in a row of  $D_2$ , in which case  $T$  gets excluded in step 4. Hence  $T \notin \mathcal{T}(D)$ . By symmetry if  $T \notin \mathcal{T}(D_1)$  then  $T \notin \mathcal{T}(D)$ . We conclude that  $\mathcal{T}(D_1) \cap \mathcal{T}(D_2) = \mathcal{T}(D)$ .  $\square$

#### 4.4.2 Forcing Characters to be included

Below (section 4.7) we describe the application of the *Hunting for Trees* algorithms to the “Out of Africa” human mtDNA data set. In the second part of the analysis (section 4.7.4) we chose 32 sequences widely spread in all of the published phylogenies. The maximum weight tree returned was consistent with the “Africa49” hypothesis of [90], except for the placement of the Naron sequence. An examination of the data revealed that there was a character that placed the Naron sequence together with the !Kung sequences, but that there was no maximum weight tree containing the corresponding split. This motivated the question, ‘what is the maximum weight tree that contains that split?’

INSTANCE: Collection  $S$  of splits on leaf set  $L$ . Number  $d \geq 3$ . Weighting function  $w : S \rightarrow \mathbb{R}$ . Number  $d \geq 3$ . Split  $S_i \in S$ .

PROBLEM: Determine if there is a subcollection  $S' \subseteq S$  such that  $S_i \in S'$  and  $S' = \beta(T)$  for some unrooted phylogenetic  $d$ -tree  $T$ . Count the number of such trees, and determine the trees with maximum weight.

As before we solve the equivalent rooted tree problem and then transform the solution back using  $\text{UNROOT}(T, x)$ , just like in section 4.3. The rooted problem is:

INSTANCE: Collection  $C$  of clusters on leaf set  $L$ . Number  $d \geq 2$ . Weighting function  $w : C \rightarrow \mathbb{R}$ . Number  $d \geq 3$ . Cluster  $C_i \in C$ .

PROBLEM: Determine if there is a subcollection  $C' \subseteq C$  such that  $C_i \in C'$  and  $C' = \sigma(T)$  for some rooted phylogenetic  $d$ -tree  $T$ . Count the number of such trees, and determine the trees with maximum weight.

**First approach - pruning  $D$** 

Our first method for generating the set of trees in  $\mathcal{T}(C, d)$  containing some cluster  $C_i \in C$  is to prune clusters and tuples from  $D$  to effectively ‘filter out’ those trees not containing this cluster. We will assume that  $D$  has already been constructed using CREATETABLE (Algorithm 3), pruned using COUNTTREES (Algorithm 4), and had all rows  $D[i]$  removed for which  $m[i] = 0$ .

Procedure PRUNETOINCLUDE( $D, i$ )

1. FOR  $j$  from 1 to  $K$  DO
  2.   IF  $C_j$  is incompatible with  $C_i$  THEN
  3.      $D[j] \leftarrow \emptyset$
  4.   ELSE
  5.     Remove all tuples in  $D[j]$  containing clusters incompatible with  $C_i$
  6.     IF  $C_i \subset C_j$  and  $C_i \neq C_j$  THEN remove all tuples in  $D[j]$  not containing a tuple  $C_k$  such that  $C_i \subseteq C_k$ .
  7.   END(IF-ELSE)
  8. END(FOR)
- END.

Algorithm 9 : PRUNETOINCLUDE

**Theorem 4.8** *After the completion of procedure PRUNETOINCLUDE, the GETSUBCOLLECTIONS algorithm applied to  $D$  returns  $\sigma(T)$  for an arbitrary tree  $T \in \mathcal{T}(C, d)$  such that  $C_i \in \sigma(T)$ .*

*Proof*

Choose  $T \in \mathcal{T}(D)$ . If  $T$  contains a cluster incompatible with  $C_i$ , then it will be removed in step 2. If all the clusters of  $T$  are compatible with  $C_i$ , but  $T$  does not contain  $C_i$ , then  $T$  is removed in step 6. If  $T$  contains  $C_i$  then all of the tuples generating  $T$  are unaffected by the deletions.  $\square$

The attraction of this approach is that we can now apply the existing counting and optimisation algorithms to count the number of trees in  $\mathcal{T}(C, d)$  containing  $C_i$ ,

and also determine which of these trees has maximum weight. If we want to force additional clusters to be in the trees then we can apply the algorithm to further prune the table.

### Second approach - direct

The problem with the first approach is that it involves a lot of manipulation of the data structure  $D$ . If we want to do the same calculations for all of the clusters in the input set then we have to repeatedly prune and restore  $D$ . It is possible to do the counting and optimisation directly on the original table.

First we tackle the counting problem. For each  $C_i \in C$  we calculate

$$m_{incl}[i] := |\{T \in \mathcal{T}(C, d) : C_i \in \sigma(T)\}|$$

where, as before,  $\mathcal{T}(C, d)$  is the set of  $d$ -trees with clusters in  $C$ .

We will assume, as before, that  $D$  has already been constructed using CREATE TABLE (Algorithm 3), pruned using COUNT TREES (Algorithm 4), and had all rows  $D[i]$  removed for which  $m[i] = 0$ .

Procedure COUNTINCLUDETREE( $D^*, m$ )

1. Put  $m_{incl}[i] \leftarrow 0$  for all  $i = 1, \dots, K-1$ . Put  $m_{incl}[K] \leftarrow m[K]$ .
2. FOR  $i$  from  $K$  down to 1 DO
3.   FOR  $(p_1, p_2, \dots, p_q)$  in  $D^*[i]$  DO
4.     FOR  $j$  from 1 to  $q$  DO
5.        $m_{incl}[p_j] := m_{incl}[p_j] + m[p_1]m[p_2] \dots m[p_q]m_{incl}[i]/m[i]$
6.     END(FOR)
7.   END(FOR)
8. END(FOR)
- END.

Algorithm 10 : COUNTINCLUDETREE

**Theorem 4.9** *After the completion of algorithm 10,  $m_{incl}[i]$  equals the number of rooted  $d$ -trees with clusters in  $C$  that contain the cluster  $C_i$ .*

*Proof*

Not wanting to break the habit, we proceed by induction. The theorem is clearly true when  $i = K$  since  $m_{incl}[K] = m[K]$ . Suppose that it is true when  $i = K, (K - 1), \dots, j + 1$ .

In each tree  $T \in \mathcal{T}(C, d)$  such that  $C_j \in \sigma(T)$  there is a cluster  $C_l$  that is the smallest cluster of  $T$  containing  $C_j$  but not equal to  $C_j$ . We say that  $C_l$  is the cluster covering  $C_j$  in  $T$ . Our counting strategy is to go through all the clusters  $C_l$  in  $C$  such that  $C_j \subset C_l$  and sum the number of trees  $T$  in  $T \in \mathcal{T}(C, d)$  such that  $C_j, C_l \in \sigma(T)$  and  $C_l$  is the cluster covering  $C_j$  in  $T$ .

The number of  $d$ -trees  $T$  with leaf set  $C_l$  and clusters in  $C$  such that  $C_j \in \sigma(T)$  and  $C_l$  is the covering cluster of  $C_j$  equals

$$\sum m[p_1]m[p_2] \dots m[p_q]m[j]$$

where the summation is over tuples  $\{p_1, p_2, \dots, p_q, j\}$  in  $D[l]$ . If we took the set of trees in  $\mathcal{T}(C, d)$  that contain the cluster  $C_l$  and removed all the clusters in them that are subsets of  $C_l$ , then the number of distinct trees remaining is the size of the first set divided by  $m[l]$ , which, by induction, is  $m_{incl}[l]/m[l]$ . Hence the number of trees  $T$  in  $T \in \mathcal{T}(C, d)$  such that  $C_j \in \sigma(T)$  and  $C_l$  is the cluster covering  $C_j$  in  $T$  is equal to

$$\left( \sum m[p_1]m[p_2] \dots m[p_q]m[j] \right) m_{incl}[l]/m[l]$$

where the summation is over the tuples  $\{p_1, \dots, p_q, j\}$  in  $D[l]$ . To calculate  $m_{incl}[j]$  we need only take the sum of this formula over all clusters  $C_l \supset C_j$ , the approach taken by the algorithm.  $\square$

The optimisation problem is solved in a similar manner. For each  $i = 1, 2, \dots, K$  we calculate  $M_{incl}[i]$ , the maximum weight of a tree in  $\mathcal{T}(C, d)$  that contains the cluster  $C_i$ . We assume that the table  $M$  has already been constructed using the procedure MAXWEIGHTTREE (Algorithm 6).

Procedure MAXINCLUDETREE( $D, M$ )

1. Put  $M_{incl}[i] \leftarrow -\infty$  for all  $i = 1, \dots, K-1$ . Put  $M_{incl}[K] \leftarrow M[K]$ .
  2. FOR  $i$  from  $K$  down to 1 DO
  3.   FOR  $\{p_1, p_2, \dots, p_q\}$  in  $D[i]$  DO
  4.     FOR  $j$  from 1 to  $q$  DO
  5.        $M[p_j] \leftarrow \max\{M[p_j], M_{incl}[i] - M[i] + w(C_i) + \sum_{k=1}^q M[p_k]\}$
  6.     END(FOR)
  7.   END(FOR)
  8. END(FOR)
- END.

Algorithm 11 : MAXINCLUDETREE

#### 4.4.3 Triples, fans and quartets in common

Using a dynamic programming strategy we can retrieve the rooted triples and fans common to all the trees in  $\mathcal{T}(C, d)$  without having to explicitly construct the set  $\mathcal{T}(C, d)$ . This bypass is necessary because the set  $\mathcal{T}(C, d)$  can sometimes be exponentially large.

##### The rooted case

For  $i = 1, 2, \dots, K$  such that  $D[i] = \emptyset$  put

$$R[i] := \emptyset$$

and when  $D[i] \neq \emptyset$  put

$$R[i] := \bigcap_{\{p_1, \dots, p_q\} \in D[i]} (r(C_{p_1}, C_{p_2}, \dots, C_{p_q}) \cup R[p_1] \cup \dots \cup R[p_q])$$

where  $r(X_1, X_2, \dots, X_k)$  for a collection of disjoint clusters  $X_1, \dots, X_k$  equals the set of rooted triples

$$\bigcup_{i \neq j} \{ab|c : a, b \in X_i, c \in X_j\}.$$

Likewise, for all  $i = 1, \dots, K$  such that  $D[i] = \emptyset$  put

$$F[i] := \emptyset$$

and when  $D[i] \neq \emptyset$  put

$$F[i] := \bigcap_{\{p_1, \dots, p_q\} \in D[i]} f(C_{p_1}, \dots, C_{p_q}) \cup F[p_1] \cup \dots \cup F[p_q]$$

where  $f(X_1, X_2, \dots, X_k)$  for a collection of disjoint clusters  $X_1, \dots, X_k$  equals the set of fans

$$\bigcup_{\{i,j,k\} \subseteq \{1, \dots, k\}} \{(a, b, c) : a \in X_i, b \in X_j, c \in X_k\}.$$

**Theorem 4.10** For  $i = 1, \dots, K$  let  $\mathcal{T}_i$  be the set of trees with leaf set  $C_i$  and clusters in  $C = \{C_1, \dots, C_K\}$ . Then

$$R[i] = \bigcap_{T \in \mathcal{T}_i} r(T_i)$$

and

$$F[i] = \bigcap_{T \in \mathcal{T}_i} f(T_i).$$

*Proof*

We prove the result by induction on  $i = 1, \dots, K$  (yawn). If  $C_i$  has only one element then the result is trivial. Assume the result holds for  $i = 1, 2, \dots, j-1$ .

Let  $\{p_1, p_2, \dots, p_q\} \in D[j]$ . There is a tree  $T \in \mathcal{T}_j$  such that the maximal subtrees of  $T$  have leaf sets  $C_{p_1}, C_{p_2}, \dots, C_{p_q}$ .

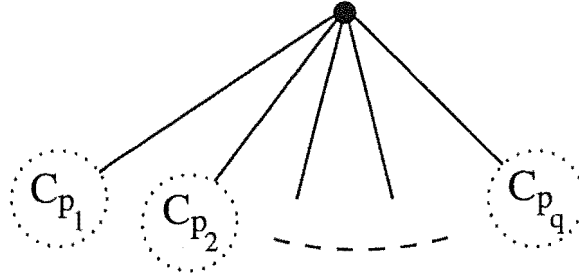


Figure 27 : The tree  $T$  that has maximal subtrees with leaf sets  $C_{p_1}, C_{p_2}, \dots, C_{p_q}$ .



Let  $\mathcal{T}_j(p_1, p_2, \dots, p_q)$  be the set of such trees in  $\mathcal{T}_j$ . A tree  $T$  is in  $\mathcal{T}_j(p_1, p_2, \dots, p_q)$  if and only if the subtree of  $T$  with leaf set  $C_{p_1}$  is in  $\mathcal{T}_{p_1}$  and the subtree of  $T$  with leaf set  $C_{p_2}$  is in  $\mathcal{T}_{p_2}$  and so on. Call these subtrees  $T_{p_1}, \dots, T_{p_q}$ . It follows that

$$\begin{aligned} \bigcap_{T \in \mathcal{T}_j(p_1, p_2, \dots, p_q)} r(T) &= \bigcap_{T \in \mathcal{T}_j(p_1, p_2, \dots, p_q)} r(C_{p_1}, \dots, C_{p_q}) \cup r(T_{p_1}) \cup \dots \cup r(T_{p_q}) \\ &= r(C_{p_1}, \dots, C_{p_q}) \cup \left( \bigcap_{S_1 \in \mathcal{T}_{p_1}} r(S_1) \right) \cup \dots \cup \left( \bigcap_{S_q \in \mathcal{T}_{p_q}} r(S_q) \right) \end{aligned}$$

from which the induction step gives

$$\bigcap_{T \in \mathcal{T}_j(p_1, p_2, \dots, p_q)} r(T) = r(C_{p_1}, \dots, C_{p_q}) \cup R[p_1] \cup \dots \cup R[p_q].$$

Likewise

$$\bigcap_{T \in \mathcal{T}_j(p_1, p_2, \dots, p_q)} f(T) = f(C_{p_1}, \dots, C_{p_q}) \cup F[p_1] \cup \dots \cup F[p_q].$$

Now for each tree  $T \in \mathcal{T}_j$  there is a unique tuple  $\{p_1, \dots, p_q\} \in D[j]$  such that  $T \in \mathcal{T}_j(p_1, \dots, p_q)$ . Hence

$$\mathcal{T}_j = \bigcup_{(p_1, \dots, p_q) \in D[j]} \mathcal{T}_j(p_1, \dots, p_q)$$

from which we deduce

$$\begin{aligned} \bigcap_{T \in \mathcal{T}_j} r(T) &= \bigcap_{(p_1, \dots, p_q) \in D[j]} \left( \bigcap_{T \in \mathcal{T}_j(p_1, p_2, \dots, p_q)} r(T) \right) \\ &= \bigcap_{(p_1, \dots, p_q) \in D[j]} r(C_{p_1}, \dots, C_{p_q}) \cup R[p_1] \cup \dots \cup R[p_q] \\ &= R[i] \end{aligned}$$

and

$$\begin{aligned} \bigcap_{T \in \mathcal{T}_j} f(T) &= \bigcap_{(p_1, \dots, p_q) \in D[j]} \left( \bigcap_{T \in \mathcal{T}_j(p_1, p_2, \dots, p_q)} f(T) \right) \\ &= \bigcap_{(p_1, \dots, p_q) \in D[j]} f(C_{p_1}, \dots, C_{p_q}) \cup F[p_1] \cup \dots \cup F[p_q] \\ &= F[i] \end{aligned}$$

as required.  $\square$

### The unrooted case

Earlier on we showed that the *Hunting for Trees* algorithm for clusters can be transformed into an algorithm for unrooted trees and splits through the use of a operations  $\text{ROOT}(S, x)$  and  $\text{UNROOT}(C, x)$ . We have just demonstrated how the rooted triples common to a the trees in  $\mathcal{T}(C, d)$  can be recovered without explicitly constructing the entire set  $\mathcal{T}(C, d)$ . Suppose we are given a set of splits  $S$  and a number  $d \geq 3$ . We would really like to use the operation  $\text{UNROOT}(C, x)$  to transform the algorithm and solve the equivalent unrooted problem, that is, recover the quartets common to all the trees in  $\mathcal{T}(S, d)$ . Unfortunately this approach doesn't work, at least not without a few modifications.

Fix  $r \in L$  and put  $C = \text{ROOT}(S, r)$ . Sort  $C$  and remove duplicates, giving an ordered set  $\{C_1, \dots, C_K\}$ . Order the set of splits  $S$  in the same way, giving  $S = \{S_1, \dots, S_K\}$ . Let  $\mathcal{T}_i$  denote the set of  $d - 1$  trees with leaf set  $C_i$  and clusters in  $C$ . Given any tree  $T \in \mathcal{T}_i$  let  $T^*$  be the unrooted tree with splits  $\text{UNROOT}(\sigma(T), r)$  where  $\text{UNROOT}$  is performed with respect to the leaf set  $L$ . Thus  $T^*$  is equal to the tree  $T$  unrooted with a new leaf labelled by  $(L - C_i)$  attached to the old root. Put  $\mathcal{T}_i^* = \{T^* : \beta(T^*) = \text{UNROOT}(\sigma(T), r), T \in \mathcal{T}_i\}$ , then  $\mathcal{T}_K = \mathcal{T}(S, d)$ . If  $D[i] = \emptyset$  then put  $Q[i] = \emptyset$  otherwise put

$$Q[i] = q(S_i) \cup \bigcap_{(p_1, \dots, p_q) \in D[i]} (Q[p_1] \cup \dots \cup Q[p_q])$$

**Theorem 4.11** *Given any  $i = 1, \dots, K$ ,*

$$Q[i] := \bigcap_{T \in \mathcal{T}_i^*} q(T).$$

*Proof*

Let's do proof by induction again, this time on  $i = 1, \dots, K$ . The theorem is true for  $i = 1$ , assume true for  $i = 1, \dots, j - 1$ . In the proof of Theorem 4.10 we let  $\mathcal{T}_j(p_1, \dots, p_q)$  denote the set of trees in  $\mathcal{T}_j$  with maximal clusters  $C_{p_1}, C_{p_2}, \dots, C_{p_q}$ . The sets  $\{\mathcal{T}_j(p_1, \dots, p_q) : \{p_1, \dots, p_q\} \in D[j]\}$  partition  $\mathcal{T}_j$ .

Define

$$\mathcal{T}_j^*(p_1, \dots, p_q) = \{T^* : \beta(T^*) = \text{UNROOT}(\sigma(T), r), T \in \mathcal{T}_j(p_1, \dots, p_q)\}$$

Then the sets  $\{\mathcal{T}_j^*(p_1, \dots, p_q) : \{p_1, \dots, p_q\} \in D[j]\}$  partition  $\mathcal{T}_j^*$ .

Fix  $\{p_1, \dots, p_q\} \in D[j]$  and choose  $T^* \in \mathcal{T}_j^*(p_1, \dots, p_q)$ . Let  $T$  be the corresponding tree in  $\mathcal{T}_j(p_1, \dots, p_q)$  and let  $T_{p_1}, \dots, T_{p_q}$  be the maximal subtrees of  $T$ . There are corresponding trees  $T_{p_1}^*$  in  $\mathcal{T}_{p_1}^*$ ,  $T_{p_2}^*$  in  $\mathcal{T}_{p_2}^*$ ,  $\dots$ ,  $T_{p_q}^*$  in  $\mathcal{T}_{p_q}^*$  (Figure 28).

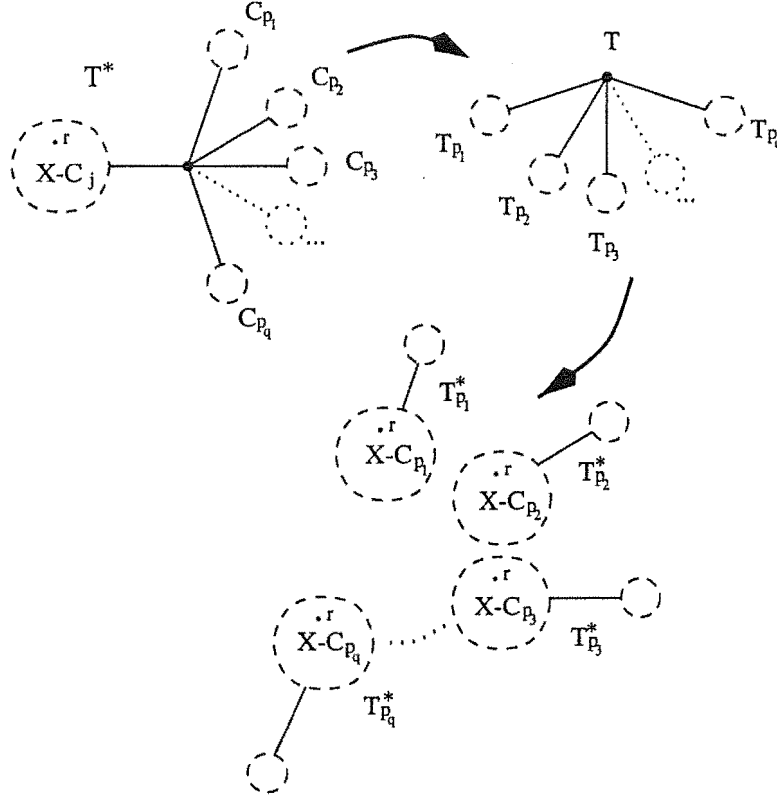


Figure 28 : We choose a tree  $T^*$  and construct the corresponding rooted tree  $T$ . The maximal subtrees of  $T$ , labelled  $T_{p_1}, \dots, T_{p_q}$ , have unrooted equivalents  $T_{p_1}^*, \dots, T_{p_q}^*$ .

Now  $\beta(T^*) = \{S_j\} \cup \beta(T_{p_1}^*) \cup \dots \cup \beta(T_{p_q}^*)$  so  $q(T^*) = q(S_j) \cup q(T_{p_1}^*) \cup \dots \cup q(T_{p_q}^*)$ . The trees in  $\mathcal{T}_j(p_1, \dots, p_q)$  are formed by replacing the subtrees  $T_{p_i}$  with any trees from  $\mathcal{T}(p_i)$ . Hence in the unrooted case

$$\bigcap_{T^* \in \mathcal{T}_j^*(p_1, \dots, p_q)} q(T^*) = q(S_j) \cup \left( \bigcap_{\tau \in \mathcal{T}_{p_1}^*} q(\tau) \right) \cup \dots \cup \left( \bigcap_{\tau \in \mathcal{T}_{p_q}^*} q(\tau) \right)$$

which by the induction step gives

$$\bigcap_{T^* \in \mathcal{T}_j^*(p_1, \dots, p_q)} q(T^*) = q(S_j) \cup Q[p_1] \cup \dots \cup Q[p_q].$$

Now

$$\begin{aligned} \bigcap_{T^* \in \mathcal{T}_j^*} q(T) &= \bigcap_{\{p_1, \dots, p_q\} \in D[j]} \bigcap_{T^* \in \mathcal{T}^*(p_1, \dots, p_q)} q(T) \\ &= q(S_j) \cup \bigcap_{\{p_1, \dots, p_q\} \in D[j]} (Q[p_1] \cup \dots \cup Q[p_q]) \end{aligned}$$

as required.  $\square$

#### 4.4.4 Automatic consensus trees

Often a tree-building method returns more than one tree; sometimes lots and lots of trees. When this happens the standard remedy is using consensus methods to find a representative tree. Consensus methods are discussed more fully in Chapter 6.

We have seen (above, Theorem 4.2) that the *Hunting for Trees* method sometimes produces an exponentially large number of solutions, even when we restrict our attention to maximum weight trees. But we needn't lose heart. We can actually calculate the strict consensus tree, the majority rule tree, and various maximum agreement subtrees without having to explicitly construct the exponentially large set of solution trees.

The method for calculating the strict consensus tree and the majority rule tree follows from the counting algorithms in section 4.2.2. The number of trees in  $\mathcal{T}(C, d)$  equals  $m[K]$ , where  $C_K$  is the cluster containing all the leaves. The strict consensus tree of the trees in  $\mathcal{T}(C, d)$  is made up of those clusters appearing in all the trees, that is, the clusters  $C_i$  for which  $m_{incl}[i] = m[K]$ .

The majority rule tree for a profile  $\mathcal{T} = \{T_1, T_2, \dots, T_k\}$  contains those clusters that appear in more than  $k/2$  trees. It follows that the majority rule tree for  $\mathcal{T}(C, d)$  is made up of those clusters  $C_i$  such that  $m_{incl}[i] > m[K]/2$ .

In Chapter 6 we present algorithms for calculating the maximum agreement subtree (MAST), the maximum information agreement subtree (MIST) and the maximum triple set agreement subtree. The algorithms take a profile of trees as input, but then work with the set of rooted triples shared by all the trees, and the set of fans shared by all the trees.

The two sets

$$R = \bigcap_{T \in \mathcal{T}(C,d)} r(T)$$

and

$$F = \bigcap_{T \in \mathcal{T}(C,d)} f(T)$$

can be constructed directly from the decomposition table  $D$  (section 4.4.3). We can calculate the maximum agreement subtrees for  $\mathcal{T}(C, d)$  from  $R$  and  $F$  and do not need to output all the trees.

## 4.5 Application to Split Decomposition and Quartet Puzzling

### 4.5.1 Split Decomposition

Split decomposition is a technique for analysing distance data that was introduced by Bandelt and Dress in [11]. Given a distance function  $d$  on a finite set  $L$  we calculate those splits  $A|B$  of  $L$  for which

$$\alpha_{A,B} = \frac{1}{2} \min_{i,j \in A, k,l \in B} (\max\{d_{ij} + d_{kl}, d_{ik} + d_{jl}, d_{il} + d_{jk}\} - d_{ij} - d_{kl})$$

is strictly positive. The quantity  $\alpha_{A,B}$  is called the **isolation index** of  $A|B$ . Put  $n = |L|$ . There can be at most  $n^2$  such splits and these can be retrieved in polynomial time.

Bandelt and Dress showed that for any distance function the set of splits with strictly positive isolation index satisfy what is called **weak compatibility**. Three splits  $A_1|B_1, A_2|B_2$  and  $A_3|B_3$  are weakly compatible if at least one of the intersections  $A_1 \cap A_2 \cap A_3, A_1 \cap B_2 \cap B_3, B_1 \cap A_2 \cap B_3, B_1 \cap B_2 \cap A_3$  is empty [12]. A set of splits is weakly compatible if and only if every subset of three splits is weakly compatible. Given any set  $\mathcal{S}$  of weakly compatible splits we can construct a distance for which the set of splits with strictly positive isolation index equals  $\mathcal{S}$ .

It was in the context of split decomposition and weakly compatible splits that the *Hunting for Trees* algorithm originated. Given a set of weakly compatible splits  $\mathcal{S}$  how do we count the number of binary trees with splits in  $\mathcal{S}$ ? How do we extract

the binary tree with maximum summed weight? Both of these problems can be solved in polynomial time using the *Hunting for Trees* algorithms. There can, after all, be at most  $O(n^2)$  splits in  $\mathcal{S}$  [11].

As before, we have a rooted equivalent of the problem. In this case the corresponding set of clusters forms a weak hierarchy. A set of clusters  $C$  is a **weak hierarchy** if and only if for every three clusters  $X, Y, Z$  the intersection  $X \cap Y \cap Z$  equals at least one of  $X \cap Y$  or  $X \cap Z$  or  $Y \cap Z$ . It is not difficult to show that if  $r \notin L$  then  $C$  is a weak hierarchy with leaf set  $L$  if and only if  $\text{UNROOT}(C, r)$  is a set of weakly compatible splits on  $L \cup \{r\}$  [10].

In the specific instance of weakly compatible splits or weak hierarchies we need to ask if we can do better than the *Hunting for Trees* algorithm. Is it possible to find not just the maximum weight  $d$ -tree, but the maximum weight tree? The following NP-completeness result indicates that this is probably not the case.

#### MAXIMUM COMPATIBLE SUBSET OF WEAK HIERARCHY

INSTANCE: Weak hierarchy  $H$ , number  $K$ .

QUESTION: Is there a compatible subset  $C \subset H$  such that  $|C| \geq K$ ?

**Theorem 4.12** *The MAXIMUM COMPATIBLE SUBSET OF WEAK HIERARCHY is NP-complete.*

*Proof*

The problem is clearly in NP.

We transform INDEPENDENT SET on graphs without triangles (see [55]) to MAXIMUM COMPATIBLE SUBSET OF WEAK HIERARCHY using a technique closely related to the proof of the main result in [39]. Let  $G, K$  make up an arbitrary instance of INDEPENDENT SET such that  $G$  has no triangles. Let  $\{v_1, \dots, v_n\}$  be the vertices of  $G$  and label the edges of  $G$  by  $e_1, \dots, e_m$ . We create a collection of  $n$  clusters  $C_1, \dots, C_n$  on leaf set  $\{v_1, \dots, v_n\} \cup \{e_1, \dots, e_m\}$  where

$$C_i = \{v_i\} \cup \{e_j : \text{the edge } e_j \text{ is adjacent to } v_i\}.$$

No cluster contains another cluster and two clusters intersect if and only if the corresponding vertices are adjacent. Hence the clusters are incompatible if and only

if the corresponding vertices are adjacent. Given any three clusters at least two of the corresponding vertices are not adjacent so at least two of the clusters are compatible. It follows that the clusters form a weak hierarchy. Furthermore a set of vertices forms an independent set in  $G$  if and only if the corresponding set of clusters is compatible.  $\square$

A direct consequence of Theorem 4.12 is that the problem of finding a compatible subset of a given size in a weakly compatible set of splits is NP-hard .

### 4.5.2 Quartet Puzzling

There are some tree building techniques so computationally difficult that it is infeasible to apply them to more than a few taxa at a time. Statistical geometry and complex model Maximum Likelihood are two examples [106]. How can we handle large taxa sets? One strategy is to construct trees on sets of four taxa and then assemble larger trees from these quartets.

But then we encounter another problem: given a set of quartets  $Q$ , can we determine if there is a binary tree  $T$  such that  $q(T) \subseteq Q$ ? In general this problem is NP-complete (FORBIDDEN QUARTETS in Appendix A). We show that the problem can be solved in polynomial time using the *Hunting for Trees* algorithms, provided that for every set of four leaves  $a, b, c, d$  at most two of  $ab|cd, ac|bd, ad|bc$  are in  $Q$ .

Let  $Q$  be such a set of quartets. We construct the set of splits  $S_Q = \{A|B : q(A|B) \subseteq Q\}$  using an adaption of the weak hierarchies algorithm in [10].

Procedure QUARTETSPLITS( $Q$ )

1. Label  $\mathcal{L}(Q)$  as  $x_1, x_2, \dots, x_k$ .
  2.  $S_1 \leftarrow \emptyset$
  3. FOR  $i$  from 2 to  $k$  DO
  4.    $S_i \leftarrow \{\{x_i\}|\{x_0, \dots, x_{i-1}\}\} \cup$   
            $\{A \cup \{x_i\}|B : A|B \in S_{i-1} \text{ and } x_i a|b_1 b_2 \in Q \text{ for all } a \in A, b_1, b_2 \in B\}$
  5. END(FOR)
  6. RETURN  $S_Q = S_k$
- END.

## Algorithm 12 : QUARTETSPLITS

It follows from [12] that any set  $S$  of splits is weakly compatible if and only if  $\cup_{A|B \in S} q(A|B)$  contains at most two quartets on the same four leaves. The set of quartets  $S_Q = \{A|B : q(A|B) \subseteq Q\}$  must therefore be weakly compatible and each partial sets of splits  $S_1, S_2, \dots, S_k$  in the algorithm QUARTETSPLITS must also be weakly compatible. Hence the algorithm runs in polynomial time, and the size of  $S_Q$  is  $O(n^2)$ . Applying the *Hunting for Trees* algorithm to  $S_Q$  we obtain a polynomial time solution to the following problem:

INSTANCE Set of quartets  $Q$  on leaf set  $L$  such that for every  $\{a, b, c, d\} \subseteq L$  at most two of  $ab|cd, ac|bd$  and  $ad|bc$  are in  $Q$ .

QUESTION Is there a binary tree  $T$  such that  $q(T) \subseteq Q$ ?

Alternatively, for fixed  $d$  we can ask whether there is a  $d$ -tree  $T$  with  $q(T) \subseteq Q$ .

## 4.6 Quartet Compatibility

Determining compatibility of a set of quartets is an NP-complete problem [101] so we do not expect to find an efficient algorithm to solve it. Earlier, in section 2.4.3 of Chapter 2, we sketched various exponential time algorithms for the problem. The rationale behind this approach was to develop techniques that work as quickly as possible, even if they don't have polynomial time complexity.

Here we introduce another technique for determining quartet compatibility, one that takes advantage of the *Hunting for Trees* algorithms. At the end of section 2.4.2 in Chapter 2 we noted that determining whether there was a split not contradicted by a set of quartets is easy when the set of quartets is compatible. We need only check all splits with two leaves on one side. We combine this observation with the following result to generate trees that extend  $Q$ .

**Theorem 4.13** *Suppose that  $A|B \in \beta(T)$  for  $T \in \langle Q \rangle$ . If  $A|B$  is non-trivial then there are splits  $A'|B'$  and  $A''|B''$  in  $CS(Q)$  such that  $A = A' \cup A''$  and  $A' \cap A'' = \emptyset$ .*



*Proof*

We can assume that  $T$  is binary, because we could just take a binary tree in  $\langle Q \rangle$  that refines  $T$ . Let  $e$  be the edge in  $T$  corresponding to  $A|B$  and let  $v$  be the end-point of  $e$  closest to the leaves in  $A$ . The remaining two edges adjacent to  $v$  give the splits  $A'|B'$  and  $A''|B''$  as required.  $\square$

The following algorithm constructs a decomposition table  $D$  such that the set of trees  $\mathcal{T}(D)$  that can be extracted from  $D$  equals the set of binary trees in  $\langle Q \rangle$ . Let  $L$  be the leaf set of  $Q$  and put  $n = |L|$ . We build up a list of clusters  $C \subseteq \text{ROOT}(CS(Q), r)$  and a decomposition table  $D$  for the clusters in  $C$ . The list  $C$  and table  $D$  are initially empty.

Procedure BUILDTREE( $Q$ )

1. Fix  $r \in L$
  2. FOR  $x$  in  $L - \{r\}$  DO
  3.     Add the entry  $\{x\}$  to the end of  $C$
  4. END(FOR)
  5.  $i \leftarrow 2$
  6. REPEAT UNTIL  $i$  is greater than the number of entries in  $C$
  7.     FOR  $j$  from 1 to  $i - 1$  DO
  8.         IF  $C[i] \cap C[j] = \emptyset$  THEN
  9.             If  $C[i] \cup C[j] = C[k]$  for some  $k$  THEN
  10.                 Append  $\{i, j\}$  to  $D[k]$
  11.             ELSE
  12.                 Append the entry  $C[i] \cup C[j]$  to the end of  $C$  and put  $(i, j)$  in the corresponding row of  $D$ .
  13.             END(IF-ELSE)
  14.         END(IF)
  15.     END(FOR)
  16.      $i \leftarrow i + 1$
  17. END(REPEAT)
- END.

Algorithm 13 : BUILDTREE

The correctness of the algorithm is a simple consequence of Theorem 4.13. To count and extract the binary trees in  $< Q >$  we need only apply the algorithms COUNTTREES and GETSUBCOLLECTIONS to the decomposition table  $D$ .

## 4.7 Application to large data sets - human mtDNA

### 4.7.1 Framework for investigation

As an illustration we apply the above algorithms to the 135 human mitochondrial DNA sequences of Vigilant et al. [111]. It was originally argued that the data supported an African origin for *homo sapiens*. The methodology of [111] was criticised by a number of people [108, 62, 74], although Penny et al. [90] have subsequently found further support for the “Out of Africa” hypothesis. We are concerned here not with the position of the root but with the branching structure of the phylogeny.

The above analyses of the human mitochondrial DNA data all used parsimony criterion to assess trees. In contrast we assess trees by the number of characters they use from the data set, where the search is restricted to those trees with bounded vertex degree and edges all supported by characters from the input. We make only a preliminary investigation, our main purpose is to demonstrate the potential of the algorithm.

It was observed in [90] (see also [62]) that all of the most parsimonious trees found shared three properties:

1. The 16 !Kung sequences together with the Naron sequence are grouped together.
2. Sequences 1-48, together with the Naron sequence are grouped together.
3. All major lineages occur in Africa, but only a subset occur in the rest of the world

These observations are used in [90] to reject the multi-regional model of human development. We wish to examine if these properties hold for trees returned by our algorithm.

### 4.7.2 Analysis of the entire set

The original data set contains 135 mtDNA sequences, each sequence being 131 sites long. We discarded those characters with missing entries. Appending the trivial splits and discarding the nine characters with three or four states, we obtained a set of 192 distinct splits. It would be interesting in future to employ more sophisticated methods of extracting binary characters from DNA sequences.

A binary tree with 135 leaves contains 267 splits, so the set could not possibly contain a set of characters that defines a binary tree. Indeed we would not even expect the data to contain all the splits of any tree with a small bound on the vertex degree. This was verified when the algorithm was run on the data set. There were no trees found, even when trees with maximum vertex degree seven were searched for, indicating that the data set does not readily support highly resolved trees. Longer sequences are required before our method can be used to analyse all 135 sequences at once. We must therefore restrict our attention to smaller subsets of taxa.

### 4.7.3 Testing the placement of the !Kung sequences

To examine whether trees returned by the algorithm satisfied property 1, we used a taxa set containing the 16 !Kung sequences, the Naron sequence, and one sequence from each of the remaining ten racial groups, the sequences selected randomly within each group. If property 1 holds then we would expect the !Kung sequences to be grouped together in the maximum weight tree.

All of the characters in the reduced data set had two or fewer states. We discarded those characters with missing entries, giving a set of 108 characters, 56 of which were constant. Adding the 27 trivial characters, and removing duplicates, we obtained a set of 52 binary characters. The algorithm quickly found 18 different trees with maximum vertex degree eight and splits contained in this data set. There were no trees with maximum vertex degree seven and splits in the data set. We increased the degree bound to nine and the algorithm found a further 94 trees.

The 52 binary characters were then weighted by the number of times each character appeared in the original set of 108 characters. There was a unique maximum weight tree. When the constant characters were included, this tree contained 91 (out of 108) characters of the original set. We present this tree below (Figure 29).

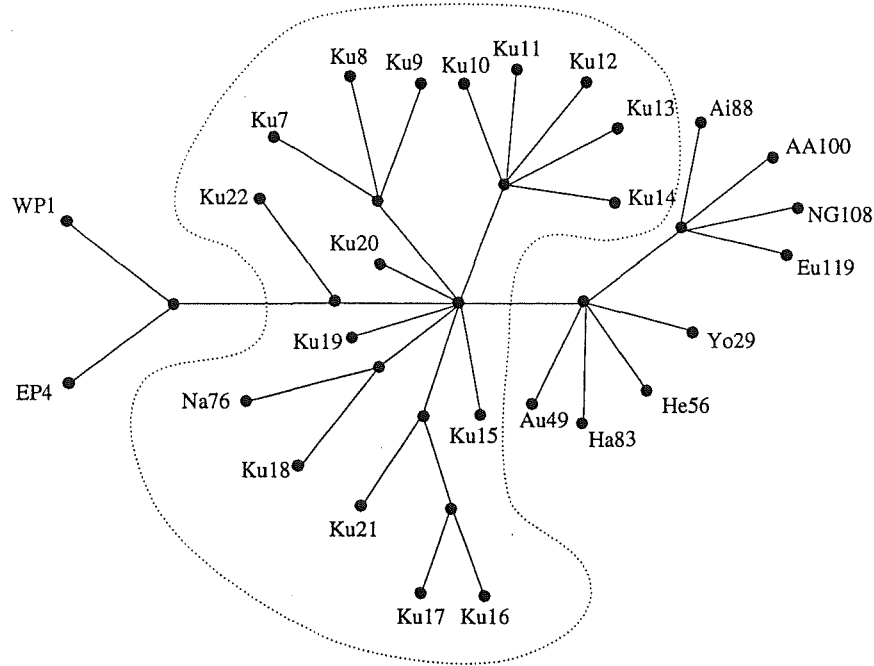


Figure 29 : The maximum weight tree for a set of mtDNA sequences containing all !Kung sequences, the Naron sequence, and one sequence from each of the remaining racial groups. The taxa are numbered as in [111]. The codes for racial groups are AA (Afroamerican), Ai (Asian), EP (Eastern Pygmy), Eu (European), Ha (Hadza), He (Herero), Ku (!Kung), Na (Naron), NG (New Guinean), WP (Western Pygmy), Yo (Yoruban).

The tree is only partially resolved, with a central vertex of degree nine. The !Kung sequences do not form a cluster in the tree but they do make up a central group.

#### 4.7.4 Testing the placement of the African sequences

We then chose a set of 32 sequences that were widely spread within each published phylogeny to test whether the maximum weight trees satisfy properties 2 and 3. Once again characters with missing sites or more than three states were discarded, the trivial characters added and duplicates removed. After several hours of processor time, the algorithm found 516 trees with maximum vertex degree nine and splits in

the data set.

Again, there was a unique maximum weight tree. It was supported by 80 (out of 136) characters (Figure 30 below). This is a considerably smaller proportion of the data set than for the previous example, indicating a greater level of uncertainty.

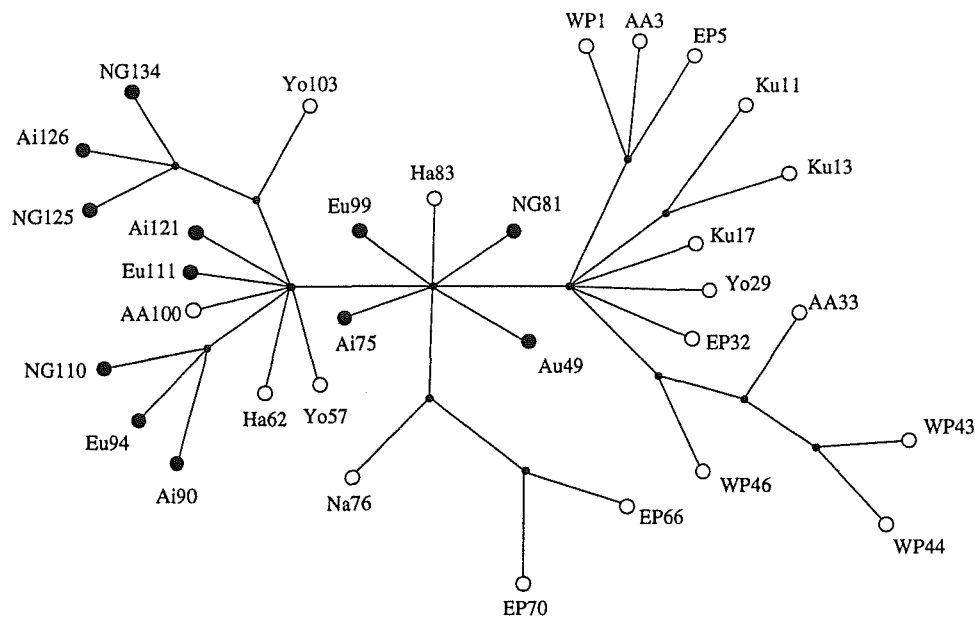


Figure 30 : The maximum weight tree for another set of mtDNA sequences. The taxa are numbered as in [111], and African sequences are marked by hollow vertices. The codes for racial groups are AA (Afroamerican), Ai (Asian), EP (Eastern Pygmy), Eu (European), Ha (Hadza), He (Herero), Ku (!Kung), Na (Naron), NG (New Guinean), WP (Western Pygmy), Yo (Yoruban).

The sequences of this set with indices less than 49 formed a cluster in this tree, so the maximum weight tree is consistent with property 2 except for the placement of the Naron sequence (Na76). There is a split in the data set (44th site) separating Ku11, Ku13, Ku17 and Na76 from the other sequences, but there is no tree with maximum vertex degree nine and splits in the data set that contains this split. It is interesting to note that the Naron sequence appears in a variety of different positions in published phylogenies (see [62, 108, 111]).

We do not have information about where to position the root in the tree so we

cannot tell what the major lineages are. However one consequence of property 3 is that the tree should have a group comprising exclusively of African sequences, a group containing both African and non-African sequences, and no group containing exclusively non-African sequences. This is clearly satisfied by the maximum weight tree in Figure 30. Hence the maximum weight tree is consistent with property 3.

It is interesting to note the differences in structure between this tree and the phylogeny of Vigilant et al. [111]. The tree in Figure 30 bears closer resemblance to the more parsimonious phylogenies given in [62] and [111].

There is considerable potential for further analysis of this data set using these methods. In particular, different character weightings and different techniques for extracting binary characters could be used. Unfortunately all analyses will be limited by the lack of resolution in the human mtDNA data set. It was this lack of resolution that forced us to use only some of sequences at a time. This problem is not unique to our approach but is faced by all studies tackling the human mtDNA data set.

# Chapter 5

## Hunting for Trees - Distance Data

### 5.1 Introduction

There are two broad categories of tree-building methods. The previous chapter was devoted to methods in the first category, those starting with discrete character data. We now venture into the second category: methods that build trees from distance data.

Distance measures are used throughout phylogenetics. They are used to estimate the similarity between two sequences or the amount of mutation needed to transform one sequence into the other. One source of data, DNA hybridization, provides distance data directly.

If we assign positive weights to the edges of a leaf labelled tree then we obtain a natural distance measure between leaves in the tree, or indeed between any two vertices in the tree. The distance between two vertices is equal to the summed weights of the edges on the path between them. Distance functions that arise this way are called additive, and can be easily characterised. Note that a **distance function** for a set  $L$  is any function  $d : L \times L \rightarrow \mathbb{R}$  such that  $d(x, y) = d(y, x)$  and  $d(x, x) = 0$ , for all  $x, y \in L$ .

The tree building problem is then, in essence, an inverse problem. Given a distance function, find the tree and set of edge weights giving a leaf to leaf distance that best approximates the input distance. There are many different criterion for assessing which approximation is ‘best’, and many more different methods for constructing possible trees.

The tree building problem divides into two parts: constructing the tree and determining the edge weights. One standard method for the second part is ordinary least squares, an approach first introduced into phylogenetics by Cavalli-Sforza and Edwards [30]. There have been various efficient algorithms proposed for this calculation [97, 110, 95]. We describe an even faster algorithm, one that is in fact time optimal.

We combine this algorithm for calculating edge weights with the *Hunting for Trees* algorithms of Chapter 4 to solve a useful variant of the Minimum Evolution Trees Problem. Given a set of splits  $S$  we can find a binary tree with the smallest minimum evolution score over all binary trees  $T$  such that  $\beta(T) \subseteq S$ . This result not only provides a tree building technique that combines distance data and character data but also leads to substantial speed-ups on local search minimum evolution tree methods.

## 5.2 Trees to Distances and Back

### 5.2.1 Three and four point conditions

Let  $T$  be a tree with leaves labelled by elements of  $L$  and edges with real valued weights. To find the distance between two leaves in the tree we sum up the weights of the edges along the unique path connecting them. We will be using  $p$  to denote leaf to leaf distance functions and  $d$  to denote general distance functions.

When the edge weights of a tree are all non-negative the resulting distance function on  $L$  is a metric and is said to be **additive**. Additive metrics can be characterised using a four point condition [28, 99]: given any four points  $a, b, c, d \in L$  we have

$$d_{ab} + d_{cd} \leq \max(d_{ac} + d_{bd}, d_{ad} + d_{bc}).$$

Suppose that there is a vertex  $v$  in  $T$  that is equidistant from all the leaves in the tree. Rooting the tree at  $v$  will give a dendrogram. Any leaf to leaf metric in such a tree is called **ultrametric**. These metrics are characterised by the three point condition:

$$d_{ac} \leq \max(d_{ab}, d_{bc})$$

where  $a, b$  and  $c$  are any three points in  $L$  [32].



There is a generalisation of additive metrics where we allow the edges in the tree to take on negative values. The resulting distance functions in this case are characterised by the simple condition that for any four points  $a, b, c, d$  at least two of

$$d_{ab} + d_{cd}, d_{ac} + d_{bd}, d_{ad} + d_{bc}$$

are equal [13]. We call this the **generalised four point condition**.

We show that an analogous result holds when we allow negative edges in the ultrametric case.

**Theorem 5.1** *A given distance function  $d$  equals the leaf to leaf distance in a dendrogram with possibly negative edge weights if and only if  $d$  satisfies the generalised four point condition and given any three points  $a, b, c$  at least two of*

$$d_{ab}, d_{ac}, d_{bc}$$

*are equal. In such a case there is a unique dendrogram with non-zero edges that realizes  $d$ .*

### *Proof*

Let  $T$  be a dendrogram with edge weights that might be negative. By considering the unrooted equivalent of  $T$  we see that the leaf to leaf distance function of  $T$  must be a generalised additive metric so therefore satisfies the generalised four point condition. Let  $v$  be the root of  $T$  and let  $\text{lca}(a, b)$  denote the lowest common ancestor of  $a$  and  $b$ . The path from  $a$  to  $v$  passes through  $\text{lca}(a, b)$  so the distance from  $a$  to  $v$  equals the distance from  $a$  to  $\text{lca}(a, b)$  plus the distance from  $\text{lca}(a, b)$  to  $v$ . The same applies for  $b$ , so we must have that the distance from  $a$  to  $\text{lca}(a, b)$  equals the distance from  $b$  to  $\text{lca}(a, b)$ . Given any three points  $a, b, c$  at least two of the lowest common ancestors of  $a$  and  $b$ , or  $a$  and  $c$ , or  $b$  and  $c$ , are equal. It follows that the three point condition holds.

Conversely, suppose that  $d$  is a distance function on a set  $L$  that satisfies the generalised four point condition and the generalised three point condition. Choose an arbitrary real number  $\lambda$  and a new leaf  $v$ . Let  $\hat{d}$  be the distance function defined on  $L \cup \{v\}$  defined by

- $\hat{d}(x, y) = d(x, y)$  for all  $x, y \in L$
- $\hat{d}(x, v) = \hat{d}(v, x) = \lambda$  for all  $x \in L$
- $\hat{d}(v, v) = 0$ .

We show that  $\hat{d}$  satisfies the generalised four point condition.

Given any four points  $a, b, c, d \in L$  at least two of  $\hat{d}_{ab} + \hat{d}_{cd}$ ,  $\hat{d}_{ac} + \hat{d}_{bd}$ ,  $\hat{d}_{ad} + \hat{d}_{bc}$  are equal, since  $\hat{d} = d$  on  $L$  and  $d$  satisfies the generalised four point condition. Given any three points  $a, b, c \in L$  we have  $\hat{d}_{ab} + \hat{d}_{cv} = \hat{d}_{ab} + \lambda$  and  $\hat{d}_{ac} + \hat{d}_{bv} = \hat{d}_{ac} + \lambda$  and  $\hat{d}_{bc} + \hat{d}_{av} = \hat{d}_{bc} + \lambda$  so at least two of these are equal since  $d$  satisfies the three point condition. The same holds if two, three or four of the four points equal  $v$ .

Since  $\hat{d}$  satisfies the generalised four point condition it equals the leaf to leaf distance of a unique tree  $T$  [13]. Root the tree  $T$  at the internal vertex adjacent to  $v$ . The leaf  $v$  is distance  $\lambda$  from all the other leaves so the internal vertex adjacent to  $v$  is equidistant to all of the leaves in  $L$ . If we remove the leaf  $v$  then we have a dendrogram which has a leaf to leaf distance function equal to  $d$ . Uniqueness follows from the uniqueness of  $T$ .  $\square$

Note that if  $d$  satisfies the generalised three point condition then it does not necessarily satisfy the generalised four point condition. Consider  $d$  defined by  $d_{ab} = d_{ac} = d_{bc} = 1$ ,  $d_{bd} = d_{cd} = d_{ad} = 2$ .

### 5.2.2 Making a space for trees

It is often convenient to represent a given distance function  $d$  by a vector in  $\mathcal{R}^{\binom{n}{2}}$ . This is the approach taken throughout the remainder of this chapter. In this representation trees correspond to subspaces of  $\mathcal{R}^{\binom{n}{2}}$ , subspaces with dimension equal to the number of edges in the tree.

Given a split  $A|B$  let  $\delta_{A|B}$  be the distance function defined by

$$\delta_{A|B}(x, y) = \begin{cases} 1 & \text{if } x \text{ and } y \text{ on different sides of the split } A|B \\ 0 & \text{otherwise} \end{cases}$$

This is called the split metric for  $A|B$  [28]. The leaf to leaf distance function  $p$  of a tree is equal to a linear combination of the split metrics corresponding to edges

in the tree:

$$p(x, y) = \sum_{A|B \in \beta(T)} \lambda_{A|B} \delta_{A|B}(x, y)$$

where the coefficients  $\lambda_{A|B}$  are the respective edge lengths [28].

In vector representation the vectors corresponding to the split metrics of a tree  $T$  span the space of leaf to leaf distance functions on  $T$ . We therefore represent a tree as a matrix  $A$ , called the **topological matrix** for  $T$ , with columns equal to the split metric vectors of splits in  $T$ . The space of leaf to leaf distance functions for  $T$  is then the span of  $A$ , and if  $b$  is a vector of edge lengths then the leaf to leaf distance function  $p$  is given by  $p = Ab$ .

Suppose that  $T$  is a tree with non-zero real weighted edges and leaf to leaf metric  $p$ . If  $wx|yz$  is a quartet, then  $wx|yz \in q(T)$  implies

$$p_{wy} + p_{xz} = p_{wz} + p_{xy}$$

To each quartet  $wx|yz$  we associate the subspace  $\mathcal{S}_{wx|yz}$  of  $\mathbb{R}^{\binom{n}{2}}$  defined by

$$\mathcal{S}_{wx|yz} = \{d : d_{wy} + d_{xz} = d_{wz} + d_{xy}\}.$$

If  $p$  is the leaf to leaf metric of  $T$  then  $wx|yz \in q(T)$  implies  $p \in \mathcal{S}_{wx|yz}$ . This leads to a necessary condition for compatibility that would be unlikely to originate from a combinatorial characterisation of the problem.

**Theorem 5.2** *If  $Q$  is a compatible set of quartets then the dimension of  $\bigcap_{wx|yz \in Q} \mathcal{S}_{wx|yz}$  is at least  $2n - 3$ .*

*Proof*

If  $Q$  is compatible then there is a binary tree  $T$  in  $\langle Q \rangle$ . If we assign any real weights to the edges of  $T$  then the resulting leaf to leaf distance function will still be contained in  $\bigcap_{wx|yz \in Q} \mathcal{S}_{wx|yz}$ .  $\square$

What if  $Q$  contains exactly one quartet for every set of four leaves  $w, x, y, z \in L$ ? Determining compatibility in this case is easy. We can also construct the Bunemann quartet tree for  $Q$ : the tree with splits  $A|B$  such that  $q(A|B) \subseteq Q$  (see section 2.4.1, page 26). This Bunemann tree arises mysteriously from our geometric characterisation:

**Theorem 5.3** *Let  $Q$  be a set of quartets such that for every set of four leaves there is exactly one quartet on that leaf set. Then*

$$\bigcap_{wx|yz \in Q} \mathcal{S}_{wx|yz}$$

*equals the subspace corresponding to the Bunemann quartet tree for  $Q$ .*

*Proof*

Let  $B_Q$  be the Bunemann quartet tree for  $Q$ .

Choose  $wx|yz \in Q$ . We wish to show that the subspace of distance functions corresponding to  $B_Q$  is contained in  $\mathcal{S}_{wx|yz}$ . Let  $p$  be a leaf to leaf distance function given by an edge weighting of  $B_Q$ . Since  $p$  is tree-like, at least two of  $p_{wx} + p_{yz}$ ,  $p_{wy} + p_{xz}$ ,  $p_{wz} + p_{xy}$  are equal. If  $p_{wy} + p_{wz} \neq p_{wz} + p_{xy}$  then either  $p_{wx} + p_{yz} = p_{wy} + p_{xz} \neq p_{wz} + p_{xy}$ , in which case  $wz|xy \in q(B_Q)$  or  $p_{wx} + p_{yz} = p_{wz} + p_{xy} \neq p_{wy} + p_{xz}$ , in which case  $wy|xz \in q(B_Q)$ . In either case we obtain a contradiction since  $q(B_Q) \subseteq Q$  and  $Q$  contains at exactly one quartet for each set of four leaves. Hence  $p_{wy} + p_{wz} = p_{wz} + p_{xy}$  and  $p \in \mathcal{S}_{wx|yz}$ .

Conversely, suppose that  $d$  is a distance function in  $\bigcap_{wx|yz \in Q} \mathcal{S}_{wx|yz}$ . Given any four points  $w, x, y, z$ , at least two of  $d_{wx} + d_{yz}$ ,  $d_{wy} + d_{xz}$ ,  $d_{wz} + d_{xy}$  are equal. Hence  $d$  is the leaf to leaf distance function of some tree  $T$  with non-zero weighted edges [13]. We want to show that  $T \sqsubseteq B_Q$  so that  $d$  is a leaf to leaf distance function for  $B_Q$  with some edge weighting.

Let  $A|B$  be a split of  $T$ . For all  $a_1, a_2 \in A$  and  $b_1, b_2 \in B$  we have  $d_{a_1b_1} + d_{a_2b_2} = d_{a_1b_2} + d_{a_2b_1} \neq d_{a_1a_2} + d_{b_1b_2}$  and so necessarily we have  $a_1a_2|b_1b_2 \in Q$ . Hence  $q(A|B) \subseteq Q$  and  $A|B \in \beta(B_Q)$ . By Theorem 1 (1) of [42],  $T \sqsubseteq T_{B_Q}$ .  $\square$

### 5.2.3 Tree building methods

It is our intention here to give a broad overview of distance based tree building methods. More detailed surveys can be found in [84, 107].

#### Exhaustive Search

To conduct an exhaustive search one generates all possible trees on a set of leaves and then evaluates each tree according to some optimality criterion. This approach

is generally not feasible due to the large number of possible trees.

### **Clustering based methods**

Many of the early methods of tree building were borrowed from cluster analysis. The basic idea is to start with  $n$  clusters, each containing a single taxa, find the two that are closest together by some criterion and combine these two clusters into one. The process is repeated until only three clusters remain, at which point the tree clusters are joined to a single vertex and the tree is constructing branching outwards from that point.

Examples of this approach are the Neighbourliness (ST) Method [97], the Neighbour Joining method (NJ) [96] and the Unweighted Pair Group Method with Arithmetic Means (UPGMA) [100] which happens to be identical to the clustering algorithm presented by Fitch and Margoliash [54].

### **Leaf addition methods**

These algorithms begin with a three leaf tree. At each step a new leaf is added in a position that optimises some criterion. Examples are the Distance Wagner Tree [51] and the modification described by Faith [48].

### **Progressive modification of distances**

The methods in this class do not build up intermediary trees during construction. They use a measure of how additive or tree-like the distance function is then progressively perturbate the function to improve this measure. The measure itself can be taken, for example, to be the amount that the distance function violates the four point condition [56].

#### **5.2.4 Optimisation Criterion**

There is little point designing a method for tree building without having some measure of how ‘good’ the resulting tree is. We describe three such criteria.

### Closest Tree

One common optimisation criterion is to measure the error, or discrepancy, between the distances in the tree and the original distance function. There are several measures available. Swofford *et al.* [106] describe four standard measures that all have the form

$$E = \sum_{i < j} w_{ij} |d_{ij} - p_{ij}|^\alpha$$

where  $d$  is the original distance,  $p$  is the distance in the tree,  $w$  is some weighting function, and  $\alpha = 1$  or  $2$ .

When  $w_{ij} = 1$  and  $\alpha = 2$  we get ordinary least squares. Day [37] showed that this optimisation problem was NP-hard, as well as the other standard variants.

### Minimum Evolution

Once we know the tree we can apply several standard algorithms to calculate optimal edge lengths. The most common are ordinary least squares (OLS), weighted least squares (WLS) and generalised least squares (GLS), all of which we discuss in section 5.3.

The Minimum Evolution method presumes that we have selected a method for approximating edge length. The **Minimum Evolution score** or **ME score** of a particular tree is then the sum of the edge weights calculated. The aim is to find the tree with the smallest score.

Minimum evolution has recently gained popularity through work by Rzhetsky and Nei [94, 95] though the criterion was actually introduced by Cavolli-Sforza and Edwards [30] (see [106]). The score in [30] equalled the sum of the absolute values of the edge weights in the tree.

### Simulation criterion

By far the most popular means to assess a method is by simulation tests. A ‘true’ tree is chosen and then distance data is generated from this tree, usually by some form of stochastic process. The various tree building methods are assessed by how often they return the true tree, and how much the reconstructed trees differ from the true tree.

## 5.3 Estimating Edge Lengths

The problem of finding the additive distance function that minimises the sum of squares error between the tree distance function and a given distance function is NP-hard [37]. However, if we are already given the tree to use, then the optimal edge weights can be found in polynomial time. This applies not only to ordinary least squares, but weighted least squares and generalised least squares criterion.

The standard approach for finding closest trees or minimum evolution trees is to construct all trees, estimate the edge lengths for each one, and then choose the closest tree or shortest tree respectively. This is somewhat impractical with more than a small number of taxa, so people usually just evaluate a large subset of trees. In any case it is important that the edge estimation technique runs as fast as possible. We describe here  $O(n^3)$  time algorithms for the ordinary least squares and weighted least squares methods and an optimal time  $O(n^4)$  algorithm for generalised least squares where the inverse of the covariance matrix  $V$  is calculated during pre-processing. In section 5.4 we describe an even faster  $O(n^2)$  time algorithm for ordinary least squares estimation.

### 5.3.1 Estimation criteria

#### Ordinary Least Squares (OLS)

The problem: we are given an unrooted tree  $T$  with leaf set  $L$  and we want to assign weights to the edges of  $T$  so that the leaf to leaf metric of  $T$  most closely approximates a given metric  $d$ , the measure of approximation being the sum of squares distance. In terms of our earlier notation (section 5.2.2), we want to find  $b$  that minimises

$$(Ab - d)^T(Ab - d)$$

where the elements of  $b$  may take on negative values.

The problem was apparently first introduced, and solved, by Cavalli-Sforza and Edwards [30]. Straightforward projection theory gives the solution

$$b = (A^T A)^{-1} A^T d$$

but direct application of this formula leads to an inefficient algorithm with complexity  $O(n^4)$ . Sattath and Tversky [97] propose a more efficient method, though

leave out the details. It seems reasonable to conclude from their description that the method they used is the same as the  $O(n^3)$  method described explicitly by Rzhetsky and Nei [95]. Formulae for calculating edge lengths have also been developed by Vach [109] (see also [110]).

### Weighted Least Squares (WLS)

The weighted least squares method for calculating edge lengths involves the minimisation of the function

$$f(b) = (Ab - d)^T W (Ab - d)$$

where  $W$  is a given diagonal matrix with strictly positive entries on the diagonal and  $b$  can have negative entries [106]. The minimum is given directly by the formula

$$b = (A^T W A)^{-1} A^T W d.$$

If we use standard matrix multiplication then this vector can be calculated in  $O(n^4)$  time. We show in section 5.3.3 that this bound can be improved to  $O(n^3)$  time.

### Generalised Least Squares (GLS)

The function to be minimised when using generalised least squares is

$$f(b) = (Ab - d)^T V^{-1} (Ab - d)$$

where  $V$ , and hence  $V^{-1}$  is a strictly positive definite symmetric matrix and  $b$  can have negative entries [106]. The direct solution is

$$b = (A^T V^{-1} A)^{-1} A^T V^{-1} d.$$

Now  $V$  is an  $\binom{n}{2} \times \binom{n}{2}$  matrix so calculating  $V^{-1}$  takes  $O(n^6)$  time. It must be remembered that this calculation is performed only once for each data set, whereas the edge weight calculation is repeated for every tree assessed. Therefore we assume that this inverse has been computed during preprocessing, before the execution of the edge weights algorithm. Even without calculating the inverse the above formula for  $b$  still takes  $O(n^5)$  time to compute.



Procedure CALCULATEAD( $T, d$ )

1. Calculate  $\delta_i^T d$  directly for all external edges.
2. REPEAT UNTIL  $\delta_i^T d$  has been calculated for all  $i = 1, \dots, |\beta(T)|$
3. Choose an internal vertex  $x$  such that exactly one of the adjacent edges has *not* had its  $\delta_i^T d$  value calculated. Let  $i$  be the index of this edge and let  $j_1, \dots, j_m$  be the indices of the remaining adjacent edges. For each  $\alpha = i, j_1, j_2, \dots, j_m$  let  $C_\alpha$  be the set of leaves on the other side of edge  $\alpha$  from  $x$  (Figure 31).
- 4.

$$\delta_i^T d \leftarrow \sum_k \delta_{j_k}^T d - 2 \sum_{k < l} \sum_{a \in C_{j_k}, b \in C_{j_l}} d_{ab}$$

5. END(REPEAT)
6. END.

Algorithm 14 : CALCULATEAD

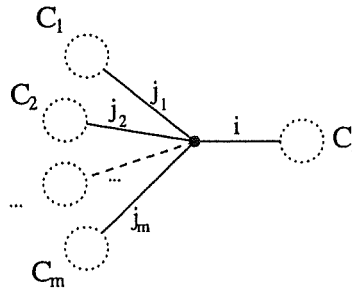


Figure 31 : Edge  $i$  with its adjacent edges and subtrees.

### 5.3.2 The first speed-up: calculating $A^T d$

Let  $T$  be a tree, not necessarily binary, and let  $A$  be the topological matrix for  $T$  (as defined on page 117). Let  $d$  be the distance function that we are trying to estimate. All of our edge length methods require the calculation of  $A^T d$ . If we use standard matrix multiplication then calculating  $A^T d$  takes  $O(n^3)$  operations, which is not surprising since  $A$  itself has  $O(n^3)$  entries. We show that  $A^T d$  can be computed in  $O(n^2)$  time when we are given the tree  $T$  as input instead of its topological matrix  $A$ .

The columns of  $A$  are equal to vectors of split metrics, denoted  $\delta_1, \dots, \delta_{|\beta(T)|}$ , so the  $i$ th entry of  $A^T d$  equals  $\delta_i^T d$  (see page 117). Procedure CALCULATEAd (Algorithm 14) calculates  $\delta_i^T d$  for all  $i = 1, \dots, |\beta(T)|$ .

We first prove that the algorithm works and then that it works in  $O(n^2)$  time. We thank Professor Andreas Dress for an observation enabling the simplification of the proof of Theorem 5.5.

**Theorem 5.4** *The algorithm CALCULATEAd correctly calculates  $\delta_i^T d$  for all  $i = 1, 2, \dots, |\beta(T)|$ .*

*Proof*

We assume that the tree  $T$  is binary since the proof for the non-binary case is essentially the same (though messier).

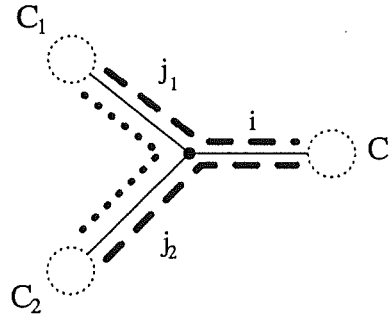


Figure 32 : A pictorial representation of the relationship between  $\delta_i^T d$ ,  $\delta_{j_1}^T d$  and  $\delta_{j_2}^T d$ . The sum  $\delta_i^T d$  is represented by the dashed lines.

The algorithm calculates the values for external edges correctly. Suppose we have the situation as in Figure 32 where the values  $\delta_{j_1}^T d$  and  $\delta_{j_2}^T d$  have been calculated correctly. Now  $\delta_i^T d$  equals the sum of all the distances going from one side of the split to the other. Hence

$$\begin{aligned} \delta_i^T d &= \sum_{a \in C_{j_1}, b \in C_i} d_{ab} + \sum_{a \in C_{j_2}, b \in C_i} d_{ab} \\ &= \sum_{a \in C_{j_1}, b \in (C_i \cup C_{j_2})} d_{ab} + \sum_{a \in C_{j_2}, b \in (C_i \cup C_{j_1})} d_{ab} - 2 \left( \sum_{a \in C_{j_1}, b \in C_{j_2}} d_{ab} \right) \\ &= \delta_{j_1}^T d + \delta_{j_2}^T d - 2 \left( \sum_{a \in C_{j_1}, b \in C_{j_2}} d_{ab} \right) \end{aligned}$$

as required.  $\square$

**Theorem 5.5** *Given a tree  $T$  and distance vector  $d$  we can calculate the vector  $A^T d$  in time  $O(n^2)$ , where  $A$  is the topological matrix of  $T$ .*

We have shown that the algorithm CALCULATEAd correctly calculates  $A^T d$ . It remains to show that the algorithm completes in  $O(n^2)$  time.

If  $\delta_i$  corresponds to a trivial split then the vector has exactly  $n - 1$  ones. Hence  $\delta_i^T d$  can be calculated in  $O(n)$  time and calculating  $\delta_i^T d$  for all the trivial splits takes  $O(n^2)$  time.

The loop in lines 2 to 5 iterates once for every edge in  $T$ , that is,  $O(n)$  times. Within iteration the sum in line 4

$$\sum_k \delta_{j_k}^T d$$

takes at most  $O(n)$  time. The second part of line 4:

$$2 \sum_{k < l} \sum_{a \in C_{j_k}, b \in C_{j_l}} d_{ab}$$

could potentially take  $O(n^2)$  time per iteration. However the total amount of time taken by this calculation over all the iterations is  $O(n^2)$ , the reason being that each individual distance  $d_{ab}$  is added at most once. Hence the entire algorithm takes  $O(n^2)$  time.  $\square$

### 5.3.3 Fast algorithms for OLS, WLS and GLS

#### Ordinary least squares

Edge lengths under OLS are given by the projection formula

$$b = (A^T A)^{-1} A^T d.$$

Using standard matrix multiplication this calculation takes  $O(n^4)$  time, where  $n$  is the number of leaves in the tree. We can speed things up using the procedure `CALCULATEAd` (Algorithm 14, page 123).

By Theorem 5.5 we can compute  $A^T d$  in  $O(n^2)$  time. For each column  $\delta_i$  of  $A$  we can calculate  $A^T \delta_i$  in  $O(n^2)$  time, so  $A^T A$  can be computed in  $O(n^3)$  time. The inversion takes  $O(n^3)$  time so the entire calculation can be completed in  $O(n^3)$  time.

We can go even faster. In section 5.4 we describe an optimal  $O(n^2)$  time algorithm.

#### Weighted least squares

Edge lengths under WLS are given by the projection formula

$$b = (A^T W A)^{-1} A^T W d.$$

Using standard matrix multiplication this calculation takes  $O(n^4)$  time. Once again, a speed-up is possible.

Working from right to left, we can calculate  $Wd$  in  $O(n^2)$  time, assuming that the input contains only the diagonal elements of  $W$ . The vector  $A^T W d$  can then be calculated in  $O(n^2)$  time using the `CALCULATEAD` algorithm.

The matrix  $(WA)$  is equal to  $A$  with the rows scaled, so can be calculated in  $O(n^3)$  time. We can then calculate  $A^T(WA)$  by applying the algorithm `CALCULATEAD` to each of the  $O(n)$  columns of  $(WA)$ , so that  $A^T W A$  is calculated in  $O(n^3)$  time. This matrix is  $n \times n$  so its inverse can be calculated in  $O(n^3)$  time. Therefore the vector  $b$  of edge lengths can be retrieved in  $O(n^3)$  time. This method works for binary and non-binary trees.

We have good reason to believe that the  $O(n^2)$  time speed-up for OLS in section 5.4 does not extend to weighted least squares. The OLS formulas takes advantage of symmetries that disappear with the introduction of a scaling matrix  $W$ .

### Generalised least squares

Edge lengths under GLS are given by the projection formula

$$b = (A^T V^{-1} A)^{-1} A^T V^{-1} d.$$

Using standard matrix multiplication this calculation takes  $O(n^5)$  time. We show how to do it in  $O(n^4)$  time.

We begin by using  $O(n^2)$  applications of CALCULATEAD to construct  $A^T V^{-1} = (V^{-1} A)^T$ . Then  $A^T V^{-1} d$  takes a further  $O(n^3)$  operations, and  $O(n^2)$  more applications of CALCULATEAD gives  $(A^T V^{-1} A)$ .

The matrix  $A^T V^{-1} A$  is of size  $n \times n$  so the inversion takes  $O(n^3)$  time, giving a total time complexity of  $O(n^4)$ . This is time optimal because there are  $O(n^4)$  entries in  $V^{-1}$ , none of which are redundant.

One useful improvement to this algorithm would be a technique that somehow bypassed the need to calculate  $V^{-1}$ . This would not, as explained above, accelerate the evaluation of each tree.

## 5.4 An unusually fast algorithm for OLS

In section 5.3.3 we described a fast algorithm for calculating edge lengths under OLS. It takes  $O(n^3)$  time, which is the same time complexity as existing algorithms [97, 109, 95]. In this section we describe an optimal  $O(n^2)$  time algorithm.

Let  $T$  be an edge weighted tree with  $n$  leaves and let  $p$  be the leaf to leaf distance function of  $T$ . Given an edge  $e_i$  with corresponding split  $A_i|B_i$ , define

$$P_i = \sum_{x \in A_i, y \in B_i} p_{xy}.$$

Thus  $P_i = \delta_i^T p$ , where  $\delta_i$  is the split metric associated with  $A_i|B_i$ .

In section 5.4.2 and section 5.4.3 we derive formulae for the length of an edge  $e_0$  in terms of  $P_0 = \delta_0^T p$ , the values  $\{\delta_i^T p : e_i \text{ is adjacent to } e_0\}$ , and the numbers of leaves in the subtrees branching off edge  $e_0$ . The formulae enable the calculation of an edge length in  $O(n)$  time. Thus  $P_i = \delta_i^T d$  and we can calculate all the edge lengths in  $O(n^2)$  time, once we know the values  $P_i$ .

Arrange the edge length formula  $b = (A^T A)^{-1} A^T d$  to give

$$A^T (Ab) = A^T d,$$

that is,

$$A^T p = A^T d$$

where  $p$  is the leaf to leaf distance function when edges are weighted according to OLS. The  $i$ th row is

$$\delta_i^T p = (A^T d)_i.$$

Hence we can calculate the values  $P_i$  in  $O(n^2)$  time using procedure CALCULATEAd (Algorithm 14).

Onward, then, to the edge formulae. First we derive matrix inversion formulae used when calculating edge lengths:

#### 5.4.1 Speeding up matrix inversion

Our fast algorithm for calculating edges under ordinary least squares requires the inversion of the matrix  $X = (nN^{-1} - 2I + U)$  where  $U$  is an  $m \times m$  matrix of ones ( $m \geq 3$ ) and  $N$  is a diagonal matrix with strictly positive integer entries that sum to  $n$ . We will prove that this matrix is always invertible and derive efficient formulae for the inverse. We solve this subproblem here so as not to interrupt the flow of the edge length derivations in the following section.

The matrix  $X$  is the sum of a diagonal matrix ( $nN^{-1} - 2I$ ) and a matrix of ones  $U$ . The inverses of such matrices, when they exist, can be completely characterised.

**Theorem 5.6** *Let  $D$  be an  $n \times n$  diagonal matrix and let  $U$  be the  $n \times n$  matrix of ones. If  $D$  has no zeros on the diagonal then put  $\kappa = 1 + \sum_{i=1}^n \frac{1}{D_{ii}}$ . The matrix  $(U + D)$  is invertible if and only if  $\kappa \neq 0$ , in which case*

$$(U + D)^{-1} = -\frac{1}{\kappa} D^{-1} U D^{-1} + D^{-1}.$$

*If  $D$  has one zero on the diagonal, say at position  $(n, n)$  then  $(U + D)$  is invertible and*

$$(U + D)^{-1} = \begin{bmatrix} 1/D_{11} & 0 & 0 & \dots & -1/D_{11} \\ 0 & 1/D_{22} & 0 & \dots & -1/D_{22} \\ 0 & 0 & \ddots & & \vdots \\ \vdots & \vdots & & & \\ -1/D_{11} & -1/D_{22} & \dots & & \kappa \end{bmatrix}$$

where  $\kappa = 1 + \sum_{i=1}^{n-1} \frac{1}{D_{ii}}$ .

If  $D$  has two or more zeros on the diagonal then  $(D + U)$  is not invertible.

*Proof*

First consider the case when  $D$  has no zeros on the diagonal, so  $D^{-1}$  exists. The  $(i, j)$  entry of  $D^{-1}UD^{-1}$  equals  $\frac{1}{D_{ii}D_{jj}}$ , so

$$\begin{aligned} (D^{-1}UD^{-1}U)_{ij} &= \frac{1}{D_{ii}} \left( \frac{1}{D_{11}} + \frac{1}{D_{22}} + \cdots + \frac{1}{D_{nn}} \right) \\ &= \frac{1}{D_{ii}} (\kappa - 1) \\ &= (\kappa - 1)(D^{-1}U)_{ij} \end{aligned}$$

and  $D^{-1}UD^{-1}U = (\kappa - 1)D^{-1}U$ .

If  $\kappa \neq 0$  then

$$\begin{aligned} \left(-\frac{1}{\kappa}D^{-1}UD^{-1} + D^{-1}\right)(U + D) &= -\frac{1}{\kappa}D^{-1}UD^{-1}U - \frac{1}{\kappa}D^{-1}U + D^{-1}U + I \\ &= -\frac{1}{\kappa}(\kappa - 1)D^{-1}U - \frac{1}{\kappa}D^{-1}U + D^{-1}U + I \\ &= \left(-1 + \frac{1}{\kappa} - \frac{1}{\kappa} + 1\right)D^{-1}U + I \\ &= I. \end{aligned}$$

However if  $\kappa = 0$  then

$$\begin{aligned} D^{-1}UD^{-1}(U + D) &= D^{-1}UD^{-1}U + D^{-1}U \\ &= (\kappa - 1)D^{-1}U + D^{-1}U \\ &= 0, \end{aligned}$$

and since  $D^{-1}UD^{-1} \neq 0$  the matrix  $(U + D)$  is not invertible.

Consider now the case when  $D$  has one zero on the diagonal. We can assume that this zero is in position  $(n, n)$  because we could otherwise just exchange rows and columns. The proof of invertibility, and of the inverse, is just a matter of multiplying the matrix given by  $(U + D)$ .

Finally, if  $D$  has more than two zeros on the diagonal then  $(U + D)$  would have two rows the same and would therefore not be invertible.  $\square$

In the specific case of the matrix  $X = (nN^{-1} - 2I + U)$  we have  $X = U + D$  where  $D = nN^{-1} - 2I$ . Let  $n_1, \dots, n_m$  be the diagonal entries of  $N$ . For the purpose of calculating edge lengths we can assume that each  $n_i$  is a strictly positive integer and  $\sum_i n_i \leq n$ . Then  $D_{ii} = 0$  if and only if  $n/n_i - 2 = 0$ , that is, if and only if  $n_i = n/2$ . Clearly this can be true for at most one  $i$ .

If  $n/n_\alpha - 2 = 0$  for some  $\alpha \in \{1, \dots, m\}$  then the matrix  $X$  is invertible by Theorem 5.6 and the inverse is given by

$$(X^{-1})_{ij} = \begin{cases} \frac{n_i}{n-2n_i} & \text{if } i = j \text{ and } i \neq \alpha \\ \kappa & \text{if } i = j = \alpha \\ \frac{-n_i}{n-2n_i} & \text{if } i \neq \alpha \text{ and } j = \alpha \\ \frac{-n_j}{n-2n_j} & \text{if } i = \alpha \text{ and } j \neq \alpha \\ 0 & \text{otherwise} \end{cases} \quad (6)$$

where  $\kappa = 1 + \sum_{i \neq \alpha} \frac{n_i}{n-2n_i}$ .

If  $n_i \neq n/2$  for all  $i = 1, \dots, m$  then we have to be a bit careful. By Theorem 5.6,  $X$  is invertible if and only if  $\kappa \neq 0$ , where  $\kappa = 1 + \sum_{i=1}^m \frac{n_i}{n-2n_i}$ . Fortunately, this is guaranteed by the following Lemma.

**Lemma 5.7** *Suppose that  $n_1, \dots, n_m$ ,  $m \geq 3$  are positive integers such that  $1 \leq n_i < n$ ,  $n_i \neq n/2$  for all  $i = 1, \dots, m$  and  $n_1 + n_2 + \dots + n_m \leq n$ . Then  $1 + \sum_{i=1}^m \frac{n_i}{n-2n_i} \neq 0$ .*

*Proof*

There is at most one  $i$  such that  $n_i > n/2$ . If there is no such  $i$  then  $\frac{n_i}{n-2n_i} > 0$  for all  $i$ , and so  $\sum_{i=1}^m \frac{n_i}{n-2n_i} + 1 > 0$ . Suppose that there is one such  $i$ . Without loss of generality we assume  $i = 1$  so  $n_1 > n/2$  and  $n_i < n/2$  for  $i = 2, \dots, m$ .

For any  $x, y$  such that  $1 \leq x, y$  and  $x + y < n/2$  we have

$$\frac{(x+y)}{n-2(x+y)} = \frac{x}{n-2(x+y)} + \frac{y}{n-2(x+y)} > \frac{x}{n-2x} + \frac{y}{n-2y}$$



so

$$\begin{aligned}
\sum_{i=1}^m \frac{n_i}{n-2n_i} + 1 &= \frac{n_1}{n-2n_1} + \sum_{i=2}^m \frac{n_i}{n-2n_i} + 1 \\
&< \frac{n_1}{n-2n_1} + \frac{\sum_{i=2}^m n_i}{n-2(\sum_{i=2}^m n_i)} + 1 \\
&= \frac{n_1}{n-2n_1} + \frac{n-n_1}{n-2(n-n_1)} + 1 \\
&= 0
\end{aligned}$$

proving the result.  $\square$

Therefore, by Theorem 5.6 we get

$$X^{-1} = -\frac{1}{\kappa}(nN^{-1} - 2I)^{-1}U(nN^{-1} - 2I)^{-1} + (nN^{-1} - 2I)^{-1}$$

where  $\kappa = 1 + \sum_{i=1}^m \frac{n_i}{n-2n_i}$ . Hence

$$(X^{-1})_{ij} = \begin{cases} -\frac{1}{\kappa} \frac{1}{(n/n_i-2)(n/n_j-2)} & \text{when } i \neq j \\ -\frac{1}{\kappa} \frac{1}{(n/n_i-2)^2} + \frac{1}{(n/n_i-2)} & \text{when } i = j \end{cases} \quad (7)$$

### 5.4.2 Internal edge length formula

Choose an internal edge  $e_0$  of the tree and let  $\alpha$  and  $\beta$  be the vertices at either end of  $e_0$  (Figure 33). Let  $e_1, \dots, e_k$  be the edges adjacent to  $e_0$  at  $\alpha$ , and let  $e_{k+1}, \dots, e_m$  be the edges adjacent at  $\beta$ . For  $i = 1, \dots, m$  let  $C_i$  be the set of leaves on the other side of  $e_i$  from  $e_0$  and put  $n_i = |C_i|$ .

Put  $n_\alpha = \sum_{i=1}^k n_i$  and  $n_\beta = \sum_{i=k+1}^m n_i$ .

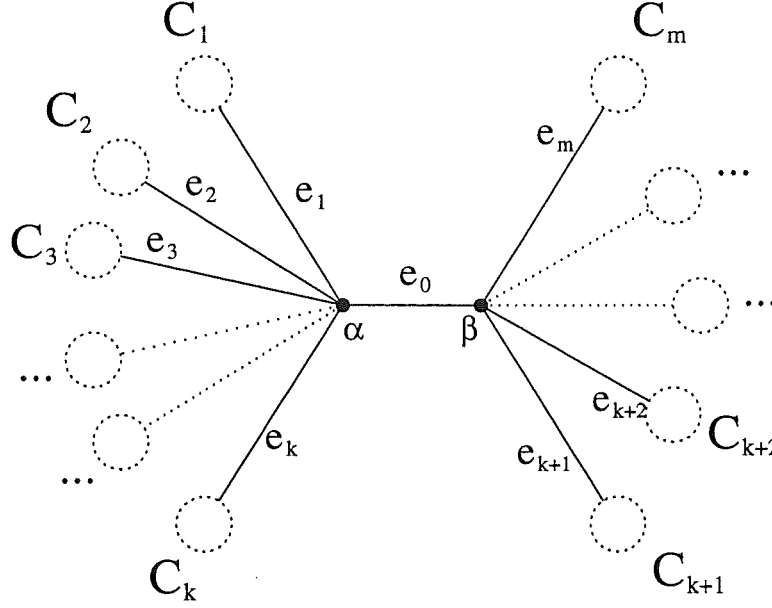


Figure 33 : The ‘generic’ internal edge with corresponding labelling.

We introduce  $m + 1$  unknowns. For  $i = 1, \dots, m$  put

$$Q_i = \begin{cases} \sum_{x \in C_i} p_{\alpha x} & i = 1, \dots, k \\ \sum_{x \in C_i} p_{\beta x} & i = k + 1, \dots, m \end{cases}$$

and let  $b_0$  be the length of the internal edge  $e_0$ . Here  $p_{uv}$  is the distance between two vertices in the tree when edges have been calculated by OLS.

The rationale behind these definitions is to define a system of linear equations that can be inverted to give a formula for the edge length  $b_0$ .

Given a split  $A_i | B_i$ ,

$$P_i = \sum_{x \in A_i, y \in B_i} p_{xy},$$

(see page 127). In the tree in Figure 33 we have

$$P_i = \sum_{j=1}^m \left( \sum_{x \in C_i, y \in C_j} p_{xy} \right) - \sum_{x, y \in C_i} p_{xy}$$

for  $i = 1, \dots, m$ . Thus we can write  $P_i$  in terms of the unknowns  $Q_1, Q_2, \dots, Q_m$  and  $b_0$ .

$$\begin{aligned}
P_i &= (n - n_i)Q_i + n_i(Q_1 + \cdots + Q_{i-1} + Q_{i+1} + \cdots + Q_m) + n_i n_\beta b_0 \\
&= (n - 2n_i)Q_i + n_i \left( \sum_{j=1}^m Q_j \right) + n_i n_\beta b_0
\end{aligned}$$

for  $i = 1, 2, \dots, k$  and

$$P_i = (n - 2n_i)Q_i + n_i \left( \sum_{j=1}^m Q_j \right) + n_i n_\alpha b_0$$

for  $i = k + 1, \dots, m$ .

More compactly we have the equation

$$\underline{P} = (nI - 2N)\underline{Q} + NU\underline{Q} + b_0 N\underline{v}$$

where  $\underline{P} = (P_1, \dots, P_m)^T$ ,  $\underline{Q} = (Q_1, \dots, Q_m)^T$ ,  $I$  is the identity matrix,  $N$  is the  $m \times m$  diagonal matrix with  $(n_1, n_2, \dots, n_m)$  on the diagonal,  $U$  is the  $m \times m$  matrix of ones, and  $\underline{v}$  is the vector with  $n_\beta$  in positions  $1, 2, \dots, k$  followed by  $n_\alpha$  in positions  $k + 1, \dots, m$ .

Rearranging we have

$$N(nN^{-1} - 2I + U)\underline{Q} = \underline{P} - b_0 N\underline{v}.$$

Put  $X = (nN^{-1} - 2I + U)$ . We proved in section 5.4.1 that  $X$  is invertible. Rearranging we obtain:

$$\underline{Q} = X^{-1}N^{-1}\underline{P} - b_0 X^{-1}\underline{v}.$$

Now consider the central branch,  $e_0$ . From the definition of  $P_i$  on page 127 we obtain

$$P_0 = \sum_{i=1}^k \sum_{j=k+1}^m \sum_{x \in C_i, y \in C_j} p_{xy}$$

which can be expressed in terms of the unknowns  $Q_1, \dots, Q_m$  and  $b_0$ .

$$\begin{aligned}
P_0 &= n_\beta(Q_1 + \cdots + Q_k) + n_\alpha(Q_{k+1} + \cdots + Q_m) + n_\alpha n_\beta b_0 \\
&= \underline{v}^T \underline{Q} + n_\alpha n_\beta b_0
\end{aligned}$$

and so

$$\begin{aligned}
n_\alpha n_\beta b_0 &= P_0 - \underline{v}^T \underline{Q} \\
&= P_0 - \underline{v}^T (X^{-1}N^{-1}\underline{P}) + b_0 \underline{v}^T X^{-1}\underline{v}
\end{aligned}$$

from which we obtain

$$\begin{aligned} b_0 &= \frac{P_0 - \underline{v}^T X^{-1} N^{-1} \underline{P}}{n_\alpha n_\beta - \underline{v}^T X^{-1} \underline{v}} \\ &= \frac{P_0 - \underline{w}^T N^{-1} \underline{P}}{n_\alpha n_\beta - \underline{w}^T \underline{v}} \end{aligned}$$

where  $\underline{w} = X^{-1} \underline{v}$ . Note that  $X$  is symmetric so  $(X^{-1})^T = X^{-1}$ .

What this formula lacks in aesthetic appeal it makes up for in utility. We show below that, using this formula, the branch length  $b_0$  can be calculated in  $O(m)$  time. But first, the external edges.

### 5.4.3 External edge length formula

Let  $e_0$  be an arbitrary external edge, let  $\alpha$  be the adjacent internal vertex, and let  $e_1, \dots, e_m$  be the other edges adjacent to  $\alpha$  (Figure 34). For each  $i = 1, \dots, m$  we let  $C_i$  denote the set of leaves on the opposite side of edges  $e_i$  from  $\alpha$  and let  $n_i$  denote the size of  $|C_i|$ .

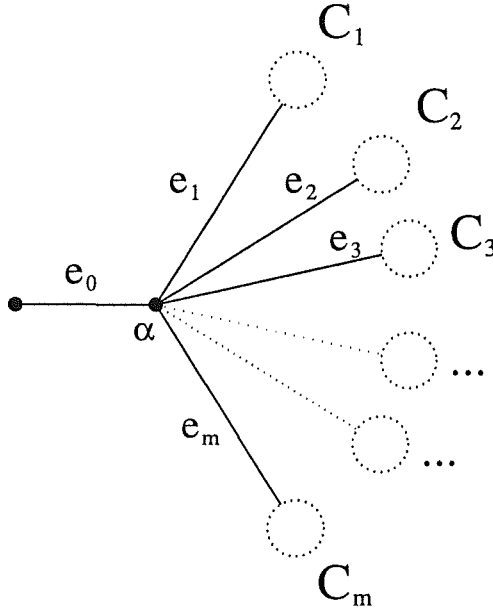


Figure 34 : The ‘generic’ external length and adjacent edges.

We introduce  $m + 1$  unknowns: for each  $i = 1, \dots, m$  put  $Q_i = \sum_{x \in C_i} p_{\alpha x}$ . Here  $p_{uv}$  is the distance between two vertices in the tree when edges have been calculated by OLS. Let  $b_0$  be the length of  $e_0$ . We build up our system of equations.

For any  $i = 1, \dots, m$  we have

$$P_i = (n - n_i)Q_i + n_i(Q_1 + \dots + Q_i + Q_{i+1} + \dots + Q_m) + b_0 n_i$$

which gives the equation

$$\underline{P} = (nI - 2N)\underline{Q} + NU\underline{Q} + b_0 N\underline{v}$$

where  $P, Q, N, I$  and  $U$  are defined as in the internal edge case and  $\underline{v}$  is a vector of  $m$  ones. Put  $X = (nN^{-1} - 2I + U)$ . We proved in section 5.4.1 that  $X$  is invertible giving:

$$\underline{Q} = X^{-1}N^{-1}\underline{P} - b_0 X^{-1}\underline{v}.$$

Looking at the external edge  $e_0$  and expanding  $P_0$  gives

$$\begin{aligned} P_0 &= (n - 1)b_0 + (Q_1 + Q_2 + \dots + Q_m) \\ &= (n - 1)b_0 + \underline{v}^T \underline{Q} \end{aligned}$$

where, once again,  $\underline{v}$  is the vector of  $m$  ones.

This is identical to the internal edge case with  $k = m$ ,  $n_\beta = 1$  and  $n_\alpha = (n - 1)$ . Performing the same manipulations gives

$$b_0 = \frac{P_0 - \underline{w}^T N^{-1} \underline{P}}{n_\alpha n_\beta - \underline{w}^T \underline{v}}$$

where  $\underline{w} = X^{-1}\underline{v}$ .

#### 5.4.4 Simplifications for binary trees

Though we can use the above results to obtain formulas for internal and external edge lengths in binary trees a modified approach gives a simpler formula. We can use a fuller set of equations. For an internal edge  $e_0$  define  $e_i, n_i$  and  $Q_i$  as before

(Figure 35). We derive the matrix equation

$$\begin{bmatrix} P_1 \\ P_2 \\ P_3 \\ P_4 \\ P_0 \end{bmatrix} = \begin{bmatrix} n_2 + n_3 + n_4 & n_1 & n_1 & n_1 & n_1(n_3 + n_4) \\ n_2 & n_1 + n_3 + n_4 & n_2 & n_2 & n_2(n_3 + n_4) \\ n_3 & n_3 & n_1 + n_2 + n_4 & n_3 & n_3(n_1 + n_2) \\ n_4 & n_4 & n_4 & n_1 + n_2 + n_3 & n_4(n_1 + n_2) \\ n_3 + n_4 & n_3 + n_4 & n_1 + n_2 & n_1 + n_2 & (n_1 + n_2)(n_3 + n_4) \end{bmatrix} \begin{bmatrix} Q_1 \\ Q_2 \\ Q_3 \\ Q_4 \\ b_0 \end{bmatrix}$$

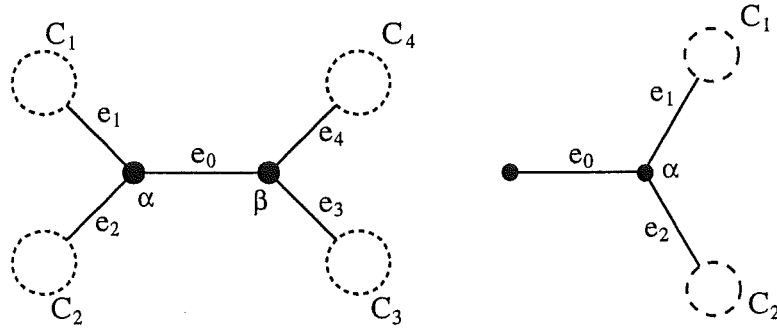


Figure 35 : The generic internal and external edges in a binary tree.

Inverting the matrix gives a closed formula for  $b_0$ :

$$b_0 = \left( \left( \frac{n}{n_4} + \frac{n}{n_3} + \frac{n}{n_2} + \frac{n}{n_1} - 4 \right) P_0 + \frac{n_1 + n_2}{n_1 n_2} ((2n_2 - n)P_1 + (2n_1 - n)P_2) + \frac{n_3 + n_4}{n_3 n_4} ((2n_4 - n)P_3 + (2n_3 - n)P_4) \right) \frac{1}{4(n_1 + n_2)(n_3 + n_4)}.$$

Similar, but simpler, arguments give a formula for external edge lengths:

$$b_0 = \frac{1}{4(n_1 n_2)} ((1 + n_1 + n_2)P_0 - (1 + n_1 - n_2)P_1 - (1 - n_1 + n_2)P_2).$$

Note that the formulas of Rzhetsky and Nei [95] can be easily recovered from these formulae by substituting

$$\begin{aligned}
P_1 &= d_{AB} + d_{AC} + d_{AD} \\
P_2 &= d_{AB} + d_{BC} + d_{BD} \\
P_3 &= d_{AC} + d_{BC} + d_{CD} \\
P_4 &= d_{AD} + d_{BD} + d_{CD} \\
P_0 &= d_{AC} + d_{BC} + d_{AD} + d_{BD}
\end{aligned}$$

bypassing the extended and sometimes tortuous derivations in [95].

#### 5.4.5 The algorithm

The simplifications for binary trees make it clear that estimating edge lengths in binary trees takes only  $O(n^2)$  time: calculate all of the values for  $P_i$  using CALCULATEAD and then apply the formulae to each edge. We show here that the same degree of complexity can be achieved in non-binary trees, thanks to the formulae developed above.

The formula

$$b_0 = \frac{P_0 - \underline{w}^T N^{-1} \underline{P}}{n_\alpha n_\beta - \underline{w}^T \underline{v}}$$

derived in section 5.4.2 and section 5.4.3 would provide an  $O(n)$  time method for calculating  $b_0$  if we could calculate  $\underline{w} = X^{-1} \underline{v}$  in  $O(n)$  time.

We adopt the labelling and notation used in section 5.4.3. Recall from section 5.4.1 that there were two cases to consider when calculating  $X^{-1}$ .

First suppose that there is some  $\alpha \in \{1, \dots, m\}$  such that  $n/n_\alpha - 2 = 0$ . If  $i \neq \alpha$  then equation 6 on page 130 gives

$$\begin{aligned}
\underline{w}_i &= (X^{-1} \underline{v})_i \\
&= \sum_{j=1}^m (X^{-1})_{ij} \underline{v}_j \\
&= \frac{\underline{v}_i - \underline{v}_\alpha}{n/n_i - 2}
\end{aligned}$$

and

$$\begin{aligned}
 \underline{w}_\alpha &= (X^{-1}\underline{v})_\alpha \\
 &= \sum_{j=1}^m (X^{-1})_{\alpha j} \underline{v}_j \\
 &= \left( \sum_{j \neq \alpha} \frac{-n_j \underline{v}_j}{n - 2n_j} \right) + \kappa \underline{v}_\alpha
 \end{aligned}$$

where  $\kappa = 1 + \sum_{i \neq \alpha} \frac{n_i}{n - 2n_i}$ . We can calculate  $\kappa$  in  $O(m)$  time and then calculate all of  $\underline{w}$  in  $O(m)$  time. Note that  $m \leq n$ .

Suppose that  $n_i \neq n/2$  for all  $i = 1, \dots, m$ . We make use of equation 7 on page 131.

$$\begin{aligned}
 w_i &= (X^{-1}v)_i \\
 &= \sum_{j=1}^m (X^{-1})_{ij} \underline{v}_j \\
 &= \sum_{j=1}^m \frac{-1}{\kappa} \frac{\underline{v}_j}{(n/n_i - 2)(n/n_j - 2)} + \frac{\underline{v}_i}{(n/n_i - 2)} \\
 &= \frac{-1}{\kappa} \frac{1}{(n/n_i - 2)} \left( \sum_{j=1}^m \frac{\underline{v}_j}{(n/n_j - 2)} \right) + \frac{\underline{v}_i}{(n/n_i - 2)} \\
 &= \frac{1}{(n/n_i - 2)} (\gamma + \underline{v}_i)
 \end{aligned}$$

where  $\kappa = 1 + \sum_{j=1}^m \frac{n_j}{n - 2n_j}$  and  $\gamma = \frac{-1}{\kappa} \sum_{j=1}^m \frac{\underline{v}_j}{(n/n_j - 2)}$ . We calculate  $\kappa$  and  $\gamma$  first, in  $O(m)$  time, and then calculate all of  $\underline{w}$  in  $O(m)$  time.

We summarise these results with an algorithm (Algorithm 15). The procedure CALCULATEEDGES takes as input: (i) a leaf labelled tree  $T$  stored as a table of edges; (ii) a distance vector  $d$  stored as an  $\binom{n}{2}$  vector, where  $n$  is the number of leaves in  $T$ . It returns a real valued weighting for each edge. Note that most representations for  $T$  can be converted into a list of edges in  $O(n^2)$  time.

Not long after completing this work I discovered that some of the OLS formulae were first discovered several years ago. In a 1989 conference paper Werner Vach [109] gave formulae for calculating edge lengths under OLS. Though Vach's notation is somewhat cryptic, it is possible to demonstrate equivalence. The contribution of the present thesis is a complete, and improved, derivation and a speed up from an  $O(n^3)$  time algorithm to an optimal time  $O(n^2)$  time algorithm.



Procedure CALCULATEEDGES( $T, d$ )

1. Calculate the number of leaves on each side of each edge.
2. Calculate  $P_i$  for each edge using CALCULATEAD.
3. FOR each edge  $e_i$  DO
4.   IF  $e_i$  is internal THEN label the edges adjacent to  $e_i$  as in Figure 33. IF  $e_i$  is external THEN label the edges adjacent to  $e_i$  as in Figure 34
5.   For each  $i$  put  $s_i = 0$  if  $n_i = n/2$  and  $s_i = n_i/(n - 2n_i)$  if  $n_i \neq n/2$ .
- 6.

$$\kappa \leftarrow 1 + \sum_{j=1}^n s_j \text{ and } \gamma \leftarrow \sum_{j=1}^k n_\beta s_j + \sum_{j=k+1}^m n_\alpha s_j$$

7.   IF  $s_i \neq 0$  for all  $i$  THEN
8.      $w_i \leftarrow -\frac{1}{n/n_i-2}(\gamma/\kappa - v_i)$
9.   ELSE find  $k$  such that  $s_k = 0$
10.     $w_i \leftarrow \frac{1}{D_{ii}}(\underline{v}_i - \underline{v}_k)$  for all  $i \neq k$
11.     $w_k \leftarrow \gamma + \kappa v_k$
12.   END(IF-ELSE)
- 13.

$$b_0 \leftarrow (P_0 - \sum_{i=1}^m \frac{w_i P_i}{n_i}) / (n_\alpha n_\beta - \sum_{i=1}^k w_i n_\beta - \sum_{i=k+1}^m w_i n_\alpha)$$

14.   Assign the edge weight  $b_0$  to the edge  $e_i$ .
15.   END(REPEAT)
- END.

Algorithm 15 : CALCULATEEDGES

## 5.5 ME trees in splits

Fast algorithms for calculating optimal edge lengths are useful, but the minimum evolution problem has two parts. Not only do we have to calculate edge weights, we have to choose the tree to calculate them on. Formally, the Minimum Evolution Tree problem is:

### MINIMUM EVOLUTION TREE

INSTANCE: Distance function  $d$  on leaf set  $L$ , number  $k$ .

QUESTION: Is there a binary tree  $T$  on  $L$  with topological matrix  $A$  such that  $\sum_i b_i < k$ , where  $b$  is the vector chosen to minimise  $\|Ab - d\|_2$ ? The vector  $b$  may have negative entries.

To our knowledge, MINIMUM EVOLUTION TREE has not been shown to be NP-complete, though it would not be too surprising in light of the depressing tendency towards intractability in this field.

There have certainly been no polynomial time algorithms for this problem. Rzhetsky and Nei present a heuristic algorithm that begins with the neighbour joining tree and then performs a local search [94]. Other approaches have been to evaluate  $\sum_i b_i$  on all possible topologies, or at best a large number of topologies. We take inspiration from the *Hunting for Trees* algorithms in Chapter 4 and modify the problem, solving the following restricted problem:

INSTANCE: Set  $S$  of splits on leaf set  $L$ . Distance function  $d$  on  $L$ .

PROBLEM: Determine if there is a binary tree  $T$  with  $\beta(T) \subseteq S$ . If there is then find a binary tree on which  $d$  gives minimum summed edge lengths under ordinary least squares.

This problem can be solved in  $O(nk + nK^3)$  time, where  $n = |L|$ ,  $k = |S|$  and  $K$  is the number of distinct splits in  $S$ .

We first describe the method used then prove that it actually works and that it has the claimed complexity.

### 5.5.1 Method

As in Chapter 4, section 4.3, we convert the problem involving unrooted trees and splits to a problem involving rooted trees and clusters. Choose an arbitrary leaf  $r$  and construct the set of clusters  $C = \text{ROOT}(S, r)$ . Apply the algorithms BUILDTABLE and COUNTTREES from Chapter 4 to construct the tables  $D$  and  $m$ . To simplify matters we delete those rows  $D[i]$  of  $D$  for which  $m[i] = 0$ .

The optimisation that we perform is done in the same way as the other *Hunting for Trees* optimisation methods but the actual value optimised requires a bit of explanation.

Given a cluster  $C_i$  we let  $\mathcal{T}_i$  equal the set of binary trees with leaf set  $C_i$  and clusters in  $C$ . Given a tree  $T \in \mathcal{T}_i$  we define  $\text{len}(T)$  as follows:

1. Take the tree  $T$ , unroot it and append a new vertex to the old root. Label the new vertex with *all* the leaves in  $L - C_i$ . This leaf will generally have more than one label (Figure 36). Call this unrooted tree  $T'$ . It has leaf set  $L$ .
2. Calculate the edge weights so that the resulting leaf to leaf distance function best approximates  $d$  under ordinary least squares.
3. Let  $\text{len}(T)$  equal the sum the weights assigned to all edges *except* the edge adjacent to the vertex labelled by  $L - C_i$ .

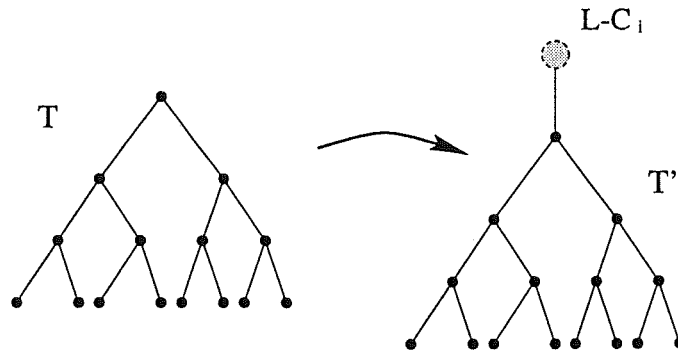


Figure 36 : The transformation from  $T$  to  $T'$ .

For each  $i = 1, 2, \dots, K$  and for each  $\{p, q\} \in D[i]$  we calculate  $val(p, q)$ , the minimum value of  $len(T)$  for all  $T \in \mathcal{T}_i$  such that  $C_p, C_q \in \sigma(T)$ . When we get to  $i = K$  we construct a binary tree with splits in  $S$  that has minimum summed weight under OLS (Algorithm 16).

The function *branchlength*, used in Algorithm 16, Algorithm 17 and Algorithm 18, is defined as follows:

If  $|C_i| = 1$  then

$$branchlength(i, j, k) := \frac{1}{4(n_j n_k)} ((1 + n_j + n_k)P_i - (1 + n_j - n_k)P_j - (1 - n_k + n_j)P_k)$$

where  $n_j = |C_j|$  and  $n_k = n - |C_k|$ . See tree (1) in Figure 37.

If  $|C_i| = n - 1$  then

$$branchlength(i, j, k) := \frac{1}{4(n_j n_k)} ((1 + n_j + n_k)P_i - (1 + n_j - n_k)P_j - (1 - n_k + n_j)P_k)$$

where  $n_j = |C_j|$  and  $n_k = |C_k|$ . See tree (2) in Figure 37.

These formulae correspond to external edge length calculations, derived on page 135.

For internal edges we define

$$branchlength(i, j, k, l, m) := \left( \left( \frac{n}{n_l} + \frac{n}{n_k} + \frac{n}{n_j} + \frac{n}{n_i} - 4 \right) P_m + \frac{n_i + n_j}{n_i n_j} ((2n_j - n)P_i + (2n_i - n)P_j) + \frac{n_k + n_l}{n_k n_l} ((2n_l - n)P_k + (2n_k - n)P_l) \right) \frac{1}{4(n_i + n_j)(n_k + n_l)}$$

where  $n_i = |C_i|$ ,  $n_j = |C_j|$ ,  $n_k = |C_k|$  and  $n_l = n - |C_l|$ . See tree (3) in Figure 37.

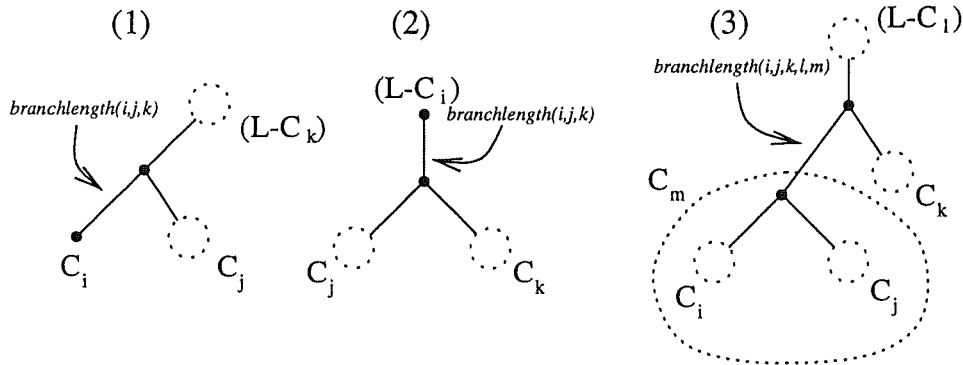


Figure 37 : The three cases in the definition of *branchlength*.

Procedure CALCULATEME( $D$ )

```

1.  FOR  $i$  from 1 to  $K$  DO
2.    IF  $|C_i| = 1$  THEN
3.       $P_i \leftarrow \sum_{x \in C_i, y \in (L - C_i)} d_{xy}$ 
4.    ELSE
5.      Choose arbitrary  $\{p, q\} \in D[i]$ 
6.       $P_i \leftarrow P_p + P_q - 2 \sum_{x \in C_p, y \in C_q} d_{xy}$ 
7.      FOR  $\{p, q\}$  in  $D[i]$  DO
8.        IF  $|C_p| = 1$  THEN
9.           $psum \leftarrow \text{branchlength}(p, q, i)$ 
10.       ELSE
11.          $psum \leftarrow \text{MIN}_{(p', q') \in D[p]} \{val(p', q') + \text{branchlength}(p', q', q, i, p)\}$ 
12.       END (IF-ELSE)
13.       IF  $|C_q| = 1$  THEN
14.          $qsum \leftarrow \text{branchlength}(q, p, i)$ 
15.       ELSE
16.          $qsum \leftarrow \text{MIN}_{(p', q') \in D[q]} \{val(p', q') + \text{branchlength}(p', q', p, i, q)\}$ 
17.       END (IF-ELSE)
18.        $val(p, q) \leftarrow psum + qsum$ 
19.     END(FOR)
20.   END(IF)
21. END(FOR)
22. RETURN  $\text{MIN}_{\{p, q\} \in D[K]} \{val(p, q) + \text{branchlength}(K, p, q)\}$ 
END.
```

Algorithm 16 : CALCULATEME

To construct a tree with weight equal to the value returned by CALCULATEME we start from the top and work down recursively.

```

Procedure EXTRACTMETREE( $D, val$ )
1. Choose  $\{p, q\}$  in  $D[K]$  that minimises
    $val(p, q) + branchlength(K, p, q)$ 
2.  $T_K \leftarrow \text{EXTRACTMESUBTREE}(D, val, \{p, q\})$ 
3. Attach the single leaf in  $L - C_K$  to the root of the  $T_K$ 
4. RETURN this tree
END.

```

Algorithm 17 : EXTRACTMETREE

Most of the work is done in the recursive procedure EXTRACTMESUBTREE (Algorithm 18).

Algorithms 17 and 18 can easily be converted into a routine that constructs *all* binary trees with splits in  $S$  and minimum sum of edge weights calculated by OLS.

### 5.5.2 Verification that the method actually works

The correctness proofs for the algorithms CALCULATEME and EXTRACTMETREE are essentially the same form as the correctness proofs for the *Hunting for Trees* algorithms in Chapter 4. The situation here is complicated by the fact that the weight assigned to an edge in a tree depends not only on the split corresponding to that edge, but also on the splits corresponding to the adjacent edges.

There are three parts to the proof. First we show that  $val(p, q) \leq len(T)$  for all  $T \in \mathcal{T}_i$  such that  $C_p, C_q \in \sigma(T)$  (Lemma 5.8(i)). Second we show that if  $T$  is the tree returned by EXTRACTMESUBTREE( $D, val, (p, q)$ ) then  $T \in \mathcal{T}_i$ ;  $C_p, C_q \in \sigma(T)$  and  $len(T) = val(p, q)$ . Finally we tie things together (Theorem 5.9) and prove that the tree returned by EXTRACTMETREE is a tree with splits in  $S$  that has minimum summed edge lengths, where the edge lengths are calculated from  $d$  by ordinary least squares.

```

Procedure EXTRACTMESUBTREE( $D, val, \{p, q\}$ )
1. IF  $|C_p| = 1$  THEN
2.   Let  $T_p$  be the single leaf in  $C_p$ 
3. ELSE
4.   Choose  $\{p', q'\}$  in  $D[p]$  that minimises
        $val(p', q') + branchlength(p', q', q, i, p)$ 
5.    $T_p \leftarrow \text{EXTRACTMESUBTREE}(D, val, (p', q'))$ 
6. END (IF-ELSE)
7. IF  $|C_q| = 1$  THEN
8.   Let  $T_q$  be the single leaf in  $C_q$ 
9. ELSE
10.  Choose  $\{p'', q''\}$  in  $D[q]$  that minimises
        $val(p'', q'') + branchlength(d, p'', q'', p, i, q)$ 
11.   $T_q \leftarrow \text{EXTRACTMESUBTREE}(D, val, (p'', q''))$ 
12. END (IF-ELSE)
13. Create a new vertex and attach it to the roots of  $T_p$  and  $T_q$ 
14. RETURN this tree
END.

```

Algorithm 18 : EXTRACTMESUBTREE

**Lemma 5.8** (i) Suppose that the algorithm CALCULATEME has completed execution. Given any  $\{p, q\} \in D[i]$  for some  $i = 1, 2, \dots, K$ , and any  $T \in \mathcal{T}_i$  such that  $C_p, C_q \in \sigma(T)$  we have  $len(T) \geq val(p, q)$ .  
(ii) If  $T$  is the tree returned by EXTRACTMESUBTREE( $D, val, (p, q)$ ) then  $len(T) = val(p, q)$

*Proof*

We use induction on  $i = 1, \dots, K$ . The result is clearly true when  $|C_p| = |C_q| = 1$ , that is, when the tree  $T$  has only two leaves. Suppose that the result is true for all  $i = 1, \dots, j - 1$ .

Choose any  $\{p, q\} \in D[j]$  and a tree  $T \in \mathcal{T}_j$  such that  $C_p, C_q \in \sigma(T)$ . Let  $T_p$  and

$T_q$  be the two maximal subtrees of  $T$ , where  $\mathcal{L}(T_p) = C_p$  and  $\mathcal{L}(T_q) = C_q$ . Construct an unrooted tree  $T'$  as follows:

1. Create a new vertex  $v$ . Unroot the trees  $T_p$  and  $T_q$  and attach the vertex  $v$  to the old root of  $T_p$  and the old root of  $T_q$ .
2. Create another vertex, labelled by the vertices in  $L - C_j$ .
3. Attach this vertex to  $v$ .

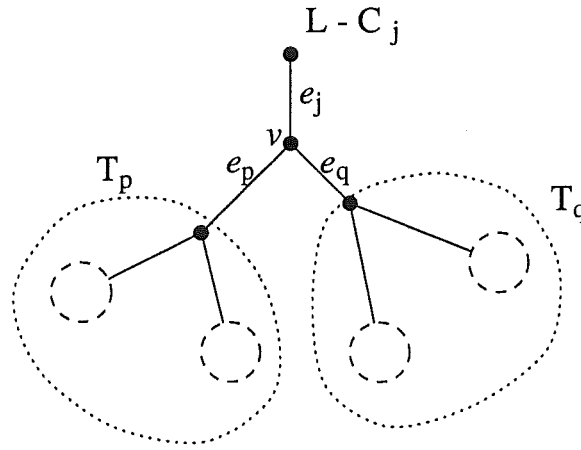


Figure 38 : The tree  $T'$  with maximal subtrees  $T_p$  and  $T_q$ .

This tree is equal to the tree constructed in the calculation of  $\text{len}(T)$  above. Let  $e_p$  be the edge in  $T'$  connecting  $v$  to the old root of  $T_p$ , let  $e_q$  be the edge connecting  $v$  and  $T_q$ , and let  $e_j$  be the edge connecting  $v$  and the vertex labelled by  $L - C_j$  (Figure 38).

In any unrooted tree, the weight assigned to an edge by OLS depends only on the input distance function, the split corresponding to the edge, and the splits corresponding to the adjacent edges. Hence all of the edge weights assigned to the edges of  $T'$  during the calculation of  $\text{len}(T)$ , except the weight assigned to edge adjacent to the vertex labelled by  $L - C_i$ , would be the same no matter how the leaves in  $L - C_i$  are resolved.



It follows from our construction of  $T'$  that

$$\text{len}(T) = \text{len}(T_p) + \text{len}(T_q) + l(e_p) + l(e_q)$$

If  $|C_p| = 1$  then

$$l(e_p) = \text{branchlength}(p, q, i).$$

The function *branchlength* is defined on page 142. Also  $T_p$  only has one vertex, so  $\text{len}(T_p) = 0$ . If  $|C_p| > 1$  then there is  $\{p', q'\} \in D[p]$  such that  $C_{p'}, C_{q'} \in \sigma(T_p)$  and so

$$l(e_p) = \text{branchlength}(p', q', q, i, p).$$

By induction  $\text{len}(T_p) \geq \text{val}(p', q')$ . Hence  $l(e_p) + \text{len}(T_p) \geq \text{psum}$  where *psum* is the quantity calculated in lines 5 and 7 of Algorithm 16.

Likewise either

$$l(e_q) = \text{branchlength}(q, p, i)$$

and  $\text{len}(T_q) = 0$ , or

$$l(e_q) = \text{branchlength}(p'', q'', p, i, q).$$

and  $\text{len}(T_q) \geq \text{val}(p'', q'')$  for some  $\{p'', q''\} \in D[q]$ . Hence  $l(e_q) + \text{len}(T_q) \geq \text{qsum}$ .

In all situations,

$$\text{len}(T) \geq l(e_p) + \text{len}(T_p) + l(e_q) + \text{len}(T_q) \geq \text{psum} + \text{qsum} = \text{val}(p, q).$$

An almost identical argument works for the proof of (ii), except that in this case when  $|C_p| > 1$  we have

$$\text{len}(T_p) + l(e_p) = \text{MIN}_{\{p', q'\} \in D[p]} \{ \text{val}(p', q') + \text{branchlength}(p', q', q, i, p) \}$$

and when  $|C_q| > 1$  we have

$$\text{len}(T_q) + l(e_q) = \text{MIN}_{\{p'', q''\} \in D[q]} \{ \text{val}(p'', q'') + \text{branchlength}(p'', q'', p, i, q) \}$$

so that  $\text{len}(T) = \text{val}(p, q)$ .  $\square$

**Theorem 5.9** *The tree returned by EXTRACTMETREE is a binary tree with splits in  $S$  that has minimum summed edge lengths, where the edge lengths are calculated from  $d$  by ordinary least squares.*

*Proof*

Let  $T'$  be any tree with splits in  $S$ . Suppose that  $r$  was the leaf used to convert the unrooted problem into a rooted problem. Let  $v$  be the leaf adjacent to the leaf  $r$ , and let  $T$  be the rooted tree obtained by removing  $r$  from  $T'$  and rooting the tree at  $v$ .

Let  $e_r$  be the edge in  $T'$  connecting the leaf  $r$  and the vertex  $v$ . The sum of the edge weights assigned to the edges of  $T'$  under OLS is equal to

$$l(e_r) + \text{len}(T)$$

There is  $\{p, q\} \in D[K]$  such that  $C_p, C_q \in \sigma(T)$ . By Lemma 5.8,  $\text{len}(T) \geq \text{val}(p, q)$ . Since the edge  $e_r$  is adjacent to a leaf and  $\{r\} = L - C_K$  we have  $l(e_r) = \text{branchlength}(K, p, q)$ . Hence the summed edge weight of  $T'$  is bounded below by

$$\text{MIN}_{(p,q) \in D[K]} \{ \text{branchlength}(K, p, q) + \text{val}(p, q) \}.$$

This lower bound is the exact summed edge weight of the tree returned by the algorithm.  $\square$

### 5.5.3 Complexity

Recall from Chapter 4, section 4.2.2 that for each  $i = 1, 2, \dots, K$  the list  $D[i]$  contains at most  $O(K)$  entries. Looking then at the algorithm CALCULATEME we see that lines 11 and 16, the two minimisation lines, can be done in  $O(K)$  time. Hence each iteration of the loop incorporating lines 8 through 18 takes  $O(K)$  time. There is one iteration for each pair  $\{p, q\} \in D[i]$  (line 3) so to complete all iterations of this loop takes  $O(K^2)$  time. Clearly the calculation of  $P_i$  in line six takes no more than  $O(n^2)$  time, which is  $O(K^2)$  since we assume  $S$  contains all the trivial splits and thus  $n \leq K$ . It follows that a single iteration of the loop incorporating lines 2 to 20 takes at most  $O(K^2)$  time. and the total time taken by the procedure CALCULATEME is  $O(K^3)$ .

If we ignore the recursion then the procedure EXTRACTMESUBTREE takes  $O(K)$  time. When we take recursion into consideration we see that the function is called only as many times as there are vertices in the final tree, that is,  $O(n)$  times. Hence extracting the tree takes at most  $O(nK)$  steps.

### 5.5.4 Application to local optimisation

An alternative to modifying intractable problems is to construct heuristic solutions. Rzhetsky and Nei describe a heuristic method for Minimum Evolution that appears to work quite well in practice [94]. First the neighbour joining tree  $T_{NJ}$  is calculated [96] and then OLS edge lengths are calculated for all binary trees within a certain distance from  $T_{NJ}$ . Here ‘distance’ means the symmetric difference distance

$$d_s(T_1, T_2) = |\beta(T_1) - \beta(T_2)| + |\beta(T_2) - \beta(T_1)|$$

(see [93, 64]). The tree with smallest summed edge lengths is then chosen as a candidate for the Minimum Evolution tree. Rzhetsky and Nei implemented this method for distances  $d_s = 2$  and  $d_s = 4$ . Note that the symmetric difference between two binary trees is always an even number.

Given any binary tree  $T$  with  $n$  leaves and fixed  $r$  there are  $O(n^{\frac{n}{2}})$  binary trees within distance  $r$  of  $T$ . Calculating the OLS edge lengths on a tree takes  $O(n^2)$  time. Therefore after the neighbor joining tree is constructed, this method takes  $O(n^{\frac{n}{2}+2})$  time to complete.

The problem is that when  $n$  is large we can only handle small values of  $r$ , and the time taken increases rapidly with  $r$ . Since  $n$  is usually much larger than  $r$ , we want an algorithm which takes polynomial time with respect to the parameter  $n$  with all the explosive, exponential time behaviour restricted to the parameter  $r$ . That is, we want to establish fixed parameter tractability for the problem [41].

We utilise the ‘ME trees in splits’ algorithm of the previous section. Given a binary tree  $T$  and  $r \geq 0$  define

$$B(T, r) = \{T' : T' \text{ binary}, d_s(T, T') \leq r\}.$$

Suppose that  $d$  is the metric for which we are trying to find a minimum evolution tree. We construct

$$BS(T, r) = \bigcup_{T' \in B(T, r)} \beta(T')$$

where in this case  $T = T_{NJ}$  the neighbour joining tree for  $d$ . The ME tree in splits algorithm is then applied to  $BS(T_{NJ}, r)$  with respect to the distance function  $d$ . This will return a binary tree with summed edge weight less than or equal to the minimum summed edge weights for all binary trees  $T'$  such that  $d_s(T_{NJ}, T') \leq r$ .

The set  $BS(T, r)$  can be easily characterised.

**Theorem 5.10**  $A|B \in BS(T, r)$  if and only if  $A|B$  is incompatible with at most  $r/2$  splits in  $\beta(T)$ .

*Proof*

Suppose  $A|B \in BS(T, r)$ . There is a binary tree  $T'$  such that  $A|B \in \beta(T')$  and  $d_s(T, T') \leq r$ . Since both  $T$  and  $T'$  are binary,  $d_s(T, T') = 2|\beta(T') - \beta(T)|$  so  $d_s(T, T') \leq r$  implies  $|\beta(T) \cap \beta(T')| \geq (2n - 3) - r/2$ . Clearly,  $A|B$  is compatible with all the splits in  $\beta(T) \cap \beta(T')$  so it is compatible with at least  $(2n - 3) - r/2$  splits in  $\beta(T)$  and is therefore incompatible with at most  $r/2$  splits in  $\beta(T)$ .

Conversely if  $A|B$  is incompatible with at most  $r/2$  splits of  $T$  then construct the set of splits of  $T$  compatible with  $A|B$ . Let  $T'$  be any binary tree containing these splits as well as  $A|B$ . Then  $|\beta(T) \cap \beta(T')| \geq (2n - 3) - r/2$  and  $d_s(T, T') \leq r$ .  $\square$

**Theorem 5.11** If  $T$  is a tree and  $A|B$  is a split then the set of edges in  $T$  corresponding to splits incompatible with  $A|B$  is connected.

*Proof*

The result is trivial if there are fewer than two splits in  $\beta(T)$  incompatible with  $A|B$ .

Let  $X$  be the set of edges in  $T$  with corresponding splits that are incompatible with  $A|B$ . Choose  $e_1$  and  $e_2$  in  $X$ . We want to show that all of the edges on the path from  $e_1$  to  $e_2$  in  $T$  are in  $X$ . The result is clearly true if  $e_1$  and  $e_2$  are adjacent. Suppose that they are not adjacent and let  $e_3$  be an edge on the path from  $e_1$  to  $e_2$ . Let  $A_1|B_1$ ,  $A_2|B_2$  and  $A_3|B_3$  be the splits corresponding to  $e_1$ ,  $e_2$  and  $e_3$ . We can write these splits so that there is  $x \in A_1 \cap A_2 \cap A_3 \cap A$ .

Since  $A|B$  is incompatible with  $A_1|B_1$  we have that  $B \cap B_1 \neq \emptyset, B, B_1$ . Likewise  $B \cap B_2 \neq \emptyset, B, B_2$ . Now  $B_2 \subset B_3 \subset B_1$  so  $B_2 \cap B \subseteq B_3 \cap B \subseteq B_1 \cap B$ . It follows that  $B_3 \cap B \neq \emptyset, B_3, B$  and  $A|B$  is incompatible with  $A_3|B_3$ . Hence  $e_3$  is in  $X$ .  $\square$

Our method for constructing  $BS(T, r)$  involves cycling through all connected subgraphs of  $T$  containing no external edges and at most  $r/2$  internal edges, generating the associated set of splits  $S$  for each subgraph and then constructing the set of splits incompatible with all of  $S$  but compatible with all of  $\beta(T) - S$ . This will also be our strategy for obtaining a rough bound on  $|BS(T, r)|$ .

**Lemma 5.12** *Let  $T$  be a binary tree with  $n$  leaves. The number of connected subgraphs of  $T$  containing no external edges and at most  $m \geq 1$  internal edges is bounded above by  $O(n4^{m-1})$ .*

*Proof*

We describe the choice of an arbitrary connected subgraph  $T'$  with at most  $m$  internal edges and no external edges. Choose an arbitrary leaf  $x$  of  $T$ .

Initially choose an internal edge for  $T'$  and label the adjacent vertex furthest away from  $x$  with 1. For each new vertex, starting with vertex 1, we decide if  $T'$  contains two more vertices adjacent to that vertex, or one more edge adjacent to that vertex (two choices), or no more edges adjacent to that vertex. Hence there are at most four possibilities for adding to each vertex. The subgraph  $T'$  contains at most  $m$  edges so we are adding to at most  $m - 1$  vertices. Hence the total number of connected subgraphs having this initial edge closest to  $x$  is bounded above by  $4^{m-1}$ . There are  $n - 3$  possible initial edges to choose from and therefore at most  $n4^{m-1}$  connected subgraphs in  $T$  containing no external edges and at most  $m$  internal edges.  $\square$

**Lemma 5.13** *Let  $T$  be a binary tree,  $E'$  a connected set of internal edges in  $T$  and  $V'$  the set of vertices adjacent to edges in  $E'$ . Let  $\kappa$  be the number of vertices in  $V'$  adjacent to at most two edges in  $E'$ . If  $S$  is the set of splits corresponding to edges in  $T$  and then the number of splits incompatible with every split in  $S$  and compatible with every split in  $\beta(T) - S$  equals  $2^{(\kappa-1)}$ .*

*Proof*

Let  $\pi$  be the partition of the  $\mathcal{L}(T)$  putting two leaves  $x$  and  $y$  in the same block if and only if the path from  $x$  to  $y$  does not pass through a vertex in  $V'$ . Construct a graph  $G$  with each vertex labelled by the leaves in a unique block in  $\pi$ . Put an edge between two vertices  $u$  and  $v$  in  $G$  if the paths in  $T$  from the leaves in the label set of  $u$  to the leaves in the label set of  $v$  contain no edges from  $E'$ . Thus  $G$  consists of isolated vertices and single edge components. The number of components of  $G$  equals  $\kappa$ .

We claim that each 2-colouring of  $G$  gives a split incompatible with all of  $S$  and compatible with all of  $\beta(T) - S$ , and conversely, each such split gives exactly two 2-colourings of  $G$ . There are exactly two ways to colour each component of  $G$ , and

since  $G$  contains  $\kappa$  components the number of different 2-colourings equals  $2^\kappa$ . The result follows.

But first we have to prove the equivalence claim.

Consider a 2-colouring of  $G$ . Construct a split  $A|B$  such that  $x \in A$  if  $x$  is in the label set of a vertex coloured white and  $x \in B$  if  $x$  is in the label set of a vertex coloured black.

Let  $A'|B'$  be a split in  $S$  and let  $e$  be the corresponding edge in  $E'$ . There are vertices  $u$  and  $v$  in  $V'$  on either side of  $e$  that are each adjacent to only one edge in  $E'$ . The vertices  $u$  and  $v$  correspond to different single edge components of  $G$ . Therefore there are leaves coloured black and leaves coloured white on both sides of the edge  $e$  and so the split  $A|B$  given by the colouration is not compatible with the split  $A'|B'$  corresponding to the edge  $e$ .

Suppose instead that  $A'|B'$  is a split in  $\beta(T) - S$ . Once again, let  $e$  be the edge of  $T$  corresponding to  $A'|B'$ . Removing  $e$  from  $T$  gives two connected components: let  $T_1$  be the one that does not contain  $V'$ . Given any two leaves  $x$  and  $y$  in  $T_1$  there is a path connecting  $x$  and  $y$  that does not pass through a vertex in  $V'$ . The two leaves must be in the same label set of a vertex in  $G$ . Hence the leaves  $x$  and  $y$ , and therefore all the leaves in  $T_1$ , are all coloured one colour and the split  $A'|B'$  is compatible with  $A|B$ .

Conversely, let  $A|B$  be a split incompatible with all the splits in  $S$  and compatible with all the splits in  $\beta(T) - S$ . Choose  $x$  and  $y$  in  $\mathcal{L}(T)$  such that the path from  $x$  to  $y$  does not contain a vertex in  $V'$ . There is at least one edge in  $T$  separating  $x$  and  $y$  from  $T'$ . Let  $e$  be one of these edges and  $A'|B'$  the split corresponding to  $e$ , where  $x, y \in A'$ . Since  $A'|B' \in \beta(T) - S$  we have  $A'|B'$  compatible with  $A|B$ .

Suppose that  $x \in A$  and  $y \in B$ . By the definition of compatibility,  $B' \cap A = \emptyset$  or  $B' \cap B = \emptyset$ . In either case,  $A|B$  would be compatible with the splits in  $S$ , a contradiction. Hence  $x$  and  $y$  are both on the same side of  $A|B$ . If two leaves are in the label set of vertex in  $G$  then they are in the same block of  $A|B$ . We can therefore colour the vertices of  $G$  so that a vertex is coloured white if the leaves in its label set are in  $A$  and black if the leaves in its label set are in  $B$ .

We have to show that this is a legitimate colouring. Let  $(u, v)$  be an edge in  $G$  and let  $U$  and  $V$  be their respective label sets. There is a vertex  $x$  in  $V'$  such that the paths from the leaves in  $U$  to the leaves in  $V$  all pass through  $x$ . Furthermore

there are edges  $e_1$  and  $e_2$  incident with  $x$  in  $T$  with associated splits  $A_1|B_1$  and  $A_2|B_2$  such that  $A_1 = U$  and  $A_2 = V$ . Let  $e_3$  be the third edge incident to  $x$ . This third edge is in  $E'$ . Let its associated split be  $A_3|B_3$ , where  $A_3 = A_1 \cup A_2$ .

By our scheme for colouring  $G$  we have that  $U \subset A$  or  $U \subset B$ , and  $V \subset A$  or  $V \subset B$ . Suppose that  $U \subset A$  and  $V \subset A$ . Then  $U \cup V \subset A$  so  $A|B$  is compatible with  $A_3|B_3$ . But this contradicts the fact that  $A_3|B_3 \in S$ . The same applies if  $U \subset B$  and  $V \subset B$ . We conclude that the  $U$  and  $V$  are on different sides of  $A|B$  and the vertices  $u$  and  $v$  are assigned a different colour.  $\square$

It is now possible to prove the complexity result.

### LOCALISED MINIMUM EVOLUTION

INSTANCE: Binary tree  $T$  on leaf set  $L$ . Distance function  $d$  on  $L$ . Integer  $r \geq 0$ .

PROBLEM Find a binary tree  $T^*$  such that the minimum evolution score of  $T^*$  is less than or equal to the minimum evolution score of  $T'$  for any binary tree  $T'$  such that  $d_s(T, T') \leq r$ .

**Theorem 5.14** *LOCALISED MINIMUM EVOLUTION can be solved in  $O(2^{(4.5r-6)}n^3)$  time, where  $n = |L|$ .*

*Proof*

The result is clearly true when  $r = 0$ . Suppose that  $r > 0$ .

We construct the set of splits  $BS(T, r)$  and then apply the ‘ME tree in splits’ algorithm of the previous section. By Theorem 5.10 each split  $A|B$  is incompatible with each split in some subset  $S \subset \beta(T)$  such that  $|S| \leq r/2$ , and by Theorem 5.11 these subset  $S$  correspond to connected subgraphs of  $T$  containing at most  $r/2$  internal edges and no external edges. The number of these subgraphs is bounded above by  $n4^{r/2-1}$ , and the number of splits compatible with the splits in  $\beta(T) - S$  and incompatible with every split in  $S$  is bounded above by  $2^{|S|} \leq 2^{r/2}$ . We must also include the  $2n - 3$  splits from  $T$ . Hence

$$\begin{aligned} |BS(T, r)| &\leq n2^{r-2}2^{r/2} + 2n - 3 \\ &= n2^{1.5r-2} + 2n - 3 \end{aligned}$$

which is  $O(n2^{1.5r-2})$ . The ‘ME Tree in Splits’ algorithm takes  $O(K^3 + nK)$  time when applied to a set of  $K$  splits. We can apply the algorithm to  $BS(T, r)$  to

solve the LOCALISED MINIMUM EVOLUTION problem in time  $O(|BS(T, r)|^3 + n|BS(T, r)|) = O(n^3 2^{4.5r-6})$  time.  $\square$

We stress that this bound is very rough and the actual time taken by the algorithm is almost definitely much smaller.



## Chapter 6

# Comparing Trees - Consensus and Agreement

### 6.1 Introduction

In a 1972 paper Edward N. Adams III presented “a new problem in the science of classification... along with its solution” [1]. The problem was how to combine information from rival trees into one representative tree, called the consensus tree. His solution came to be known as the Adams consensus tree and is only one of a multitude of possible solutions.

Ideally, all good tree building methods should produce the same tree when applied to reliable data. In practice this is seldom the case. Sometimes the different trees arise from different data sets, perhaps coming from quite different sources or different genes. On other occasions a variety of tree building methods are used on the same set of data with varying results. Some tree building methods, like maximum parsimony, may output thousands of trees and a researcher would like to assimilate this information into a single tree for interpretation. In all of these situations consensus methods are used to construct a tree best representing the conflicting classifications.

The first part of this chapter is spent surveying the theory and methods of consensus techniques. We summarise the categories of consensus methods in existence and classify them in terms of the information each method retains. We then survey developments in the axiomatic approach to consensus.

One intuitive consensus method that seems to have been overlooked in the past is the Maximum Clique Consensus tree, which we define in section 6.4. The Maximum Clique Consensus method is one of the only methods that incorporates edge weighting. Taking account of edge weights is surely a prerequisite of realistic consensus methods for edge weighted trees.

We show that, in general, constructing a Maximum Clique Consensus tree is NP-hard. However if we accept a restriction on degree bound then the *Hunting for Trees* algorithms provide polynomial time techniques.

We discuss consensus techniques based on agreement subtrees. The most popular of these is the Maximum Agreement Subtree, or MAST. We describe an algorithm for calculating a MAST tree that has the same complexity as the algorithm of [50] but is much simpler.

The MAST method has its shortcomings, as highlighted by Swofford in [107]. We show how these shortcomings can be overcome by introducing three new agreement subtree methods. We show how to construct the agreement subtree with the most edges, or the most internal edges, and also the agreement subtree containing the largest number of rooted triples or quartets. Each algorithm executes in polynomial time when applied to a profile of trees containing one tree with bounded degree.

## 6.2 A Survey of Consensus Methods

Any investigation of consensus methods should begin with a survey of the wide variety of existing consensus methods. We will too.

### 6.2.1 Strict Consensus Tree

Perhaps the simplest of the consensus methods is the strict consensus tree [2, 18, 76, 107, 116]. Given a profile of unrooted trees, the strict consensus tree contains exactly those splits common to all the trees in the profile. When the profile consists of rooted trees the strict consensus tree contains those clusters common to all the input trees.

### 6.2.2 Majority Rule Tree

Let  $\mathcal{T}$  be a profile of rooted (unrooted) trees and consider the set  $M$  of clusters (splits) contained in more than half the trees. There must be at least one tree containing any pair of these clusters (splits) so the set  $M$  is compatible. The majority rule tree, defined for rooted and unrooted trees, contains exactly those clusters (splits) in  $M$  [16, 75, 76, 107]. The majority rule tree often coincides with the median consensus tree (see below, section 6.2.8).

There are several generalisations of the majority rule method, each corresponding to different rules for selecting clusters or splits [15, 14].

### 6.2.3 Loose Consensus Tree

This tree was originally called the combinable component tree [24, 107]. We take its present name from [18]. Given a profile of unrooted trees examine every cluster of every tree in the input profile and check whether it is compatible with every tree in the profile. If it is then we include it in the loose consensus tree. Every cluster in the loose consensus tree is pairwise compatible with every cluster in every tree in the input profile.

The unrooted loose consensus tree is defined in the same way.

### 6.2.4 Adams Consensus

Adams Consensus was the first consensus method for trees and, perhaps as a consequence, is one of the most popular [1, 2, 75, 76, 107, 116]. The method is defined for rooted trees and has no analogue for unrooted trees [27]. It is perhaps most simply defined in terms of the algorithm for constructing it.

Given partitions  $\pi_1, \pi_2, \dots, \pi_k$  of the same set  $L$ , the **product partition** of  $\pi_1, \pi_2, \dots, \pi_k$  is the partition of  $L$  that puts two elements in the same block if and only if they are in the same block in all of the partitions  $\pi_1, \dots, \pi_k$ . The Adams consensus tree is constructed recursively from product partitions (Algorithm 19).

In response to the criticism that the Adams consensus tree has little intuitive justification, Adams showed that the tree preserves all nesting information common to all the input trees (see Chapter 2 section 2.3.3 for a definition of nesting). Furthermore for every pair of clusters  $X, Y$  in the Adams consensus tree we have

Procedure ADAMSTREE( $T_1, \dots, T_k$ )

1. IF the tree  $T_1$  contains only one leaf THEN return that leaf.
  2. Otherwise, for each  $i = 1, \dots, k$  let  $\pi_i$  be the partition given by the leaf sets of the maximal subtrees of  $T_i$ .
  3. Let  $\pi$  be the partition product of  $\pi_1, \dots, \pi_k$ .
  4. For each block  $B$  of  $\pi$  construct  
ADAMSTREE( $T_{1|B}, T_{2|B}, \dots, T_{k|B}$ ).
  5. Append the roots of these trees to a new vertex and RETURN this tree.
- END.

Algorithm 19 : ADAMSTREE

that  $X \subset Y$  implies  $X <_{T_i} Y$  for all trees  $T_i$  in the profile. These two properties characterise the consensus tree [2].

A consequence of the nesting property is that the Adams consensus tree also preserves the rooted triple information shared by the input trees. We show that the Adams consensus tree introduces no new rooted triple information.

**Theorem 6.1** *Let  $T^{AD}$  be the Adams consensus tree for the profile  $\mathcal{T} = \{T_1, T_2, \dots, T_k\}$ . Then*

$$r(T^{AD}) \subseteq \bigcup_i r(T_i).$$

*Proof*

Choose  $ab|c \in r(T^{AD})$ . There is some set  $S$  such that the product partition given by the maximal subtrees of  $T_{1|S}, \dots, T_{k|S}$  has  $a$  and  $b$  in one block and  $c$  in another. Hence there is some tree  $T_i$  such that  $a$  and  $b$  are in one maximal subtree of  $T_{i|S}$  and  $c$  is in another. It follows that  $ab|c \in r(T_i)$ .  $\square$

### 6.2.5 Aho *et al.* Consensus Tree

This is not really an established consensus method, though it is used in [68]. Given a profile  $\mathcal{T}$  of rooted trees construct  $R = \bigcap_{T \in \mathcal{T}} r(T)$  and then apply Algorithm 2 (ONETREE) of Chapter 2 to  $R$ .

The Aho *et al.* Consensus tree is closely linked with the Adams consensus tree.

**Theorem 6.2** *Let  $R$  be a set of rooted triples  $R$  on leaf set  $L$ . The Aho *et al.* tree for  $R$  equals the Adams Consensus tree for  $\langle R \rangle$ .*

*Proof*

Both the Aho *et al.* tree and the Adams tree are defined recursively, starting with the maximal clusters and working down towards the leaves. If we can show that the partition  $\pi_1$  given by the maximal subtrees of the Aho *et al.* tree for  $R$  is the same as the partition  $\pi_2$  given by the maximal subtrees of the Adams consensus tree for  $\langle R \rangle$  then the result follows by recursion.

The Aho *et al.* tree is in  $\langle R \rangle$  so the partition  $\pi_2$  equals the partition product of  $\pi_1$  together with all the other partitions from trees in  $\langle R \rangle$ . Hence  $\pi_2$  is a refinement of  $\pi_1$ .

If  $a$  and  $b$  are in the same block of  $\pi_1$  then they are in the same component of the graph  $[R, \mathcal{L}(R)]$ , defined on page 35. Any two leaves connected by an edge in  $[R, \mathcal{L}(R)]$  will be in the same maximal subtree of any tree in  $\langle R \rangle$ . Hence any two leaves in the same component of  $[R, \mathcal{L}(R)]$  will also be in the same maximal subtree in any tree in  $\langle R \rangle$ . Therefore  $a$  and  $b$  are in the same maximal subtree in any tree in  $\langle R \rangle$  and they are in the same block of  $\pi_2$ .

The two partitions  $\pi_1$  and  $\pi_2$  are refinements of each other, so  $\pi_1 = \pi_2$ .  $\square$

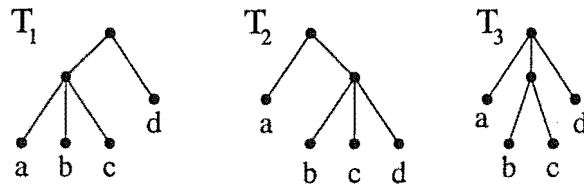


Figure 39 : The Adams consensus tree for  $T_1$  and  $T_2$  is tree  $T_3$ , but the Aho *et al.* tree for  $T_1$  and  $T_2$  is the fan tree.

Note that the Aho *et al.* tree does not equal the Adams consensus tree for every profile of trees. For example, the two trees  $T_1$  and  $T_2$  in Figure 39 have no rooted triples in common, so the corresponding Aho tree is the fan tree. However both trees have the nesting  $(b, c)a, d$ , so the Adams consensus tree for the two trees is the tree  $T_3$ .

### 6.2.6 Nelson Consensus Tree

The original definition of the Nelson consensus tree [85] exhibits a remarkable lack of rigour. The basic assumption of the method is that clusters appearing in two or more trees are likely to be genuine. These clusters are called **replicated clusters**, or **replicated components**, and the Nelson consensus tree is defined to be the tree containing the replicated clusters together with the remaining clusters that are compatible with all the replicated clusters.

There are problems with this definition. The set of replicated clusters might not be compatible, so the method does not always give a tree [76, 89, 24]. As well, there may be several distinct groups of non-replicated clusters that are compatible with the replicated clusters and with themselves, and Nelson gives no indication as to how this indeterminacy is to be resolved. “The pitfalls of philosophy are many, and some of them are deep.” [85]

Page [89] goes some way towards addressing these oversights. Each cluster in the original trees is assigned a weight equal to one less than the number of times that cluster appears in the input profile. The Nelson Consensus Tree is then taken to be the maximum weight compatible subset of this collection of clusters. If there is more than one maximum weight compatible subset then the Nelson Consensus tree equals the intersection of these maximal weight compatible sets. Note, however, that this version of the consensus tree will not contain any unreplicated clusters.

### 6.2.7 Cluster Height Methods

This class of consensus methods contains the Neumann consensus tree and its various extensions: the Durschnitt consensus tree [86], the cardinality rule consensus tree [86], and the  $s$ -consensus trees [104, 105]. Each cluster in every tree in the profile is assigned a height, and this height increases or decreases monotonically with respect

to set inclusion. For any given value  $h$  and any tree  $T_i$  in the profile there is a partition of the leaf set given by those maximal clusters with height less than  $h$  (or sometimes minimal clusters with height greater than  $h$ ). The Neumann consensus tree is constructed by taking, for each value of  $h$ , the partition product of these partitions, where the partition product is as defined for the Adams consensus tree. The variations on the Neumann consensus tree each stem from use of a different height function.

The construction of the  $s$ -consensus tree is a little more complicated. A Neumann consensus tree is built using the cardinality rule, and then clusters are removed from this tree if they have too little consensus strength, this measure being defined in [104, 105].

### 6.2.8 Median Consensus Tree

Roughly speaking, the median consensus tree represents the ‘middle ground’ between the trees in the input profile. It is based on the use of a distance measure *between* trees. The standard measure used is the symmetric difference measure, where the distance  $d_s(T_1, T_2)$  between any two unrooted trees on the same leaf set is given by

$$d_s(T_1, T_2) = |\beta(T_1) - \beta(T_2)| + |\beta(T_2) - \beta(T_1)|.$$

The distance between two rooted trees is defined by

$$d_s(T_1, T_2) = |\sigma(T_1) - \sigma(T_2)| + |\sigma(T_2) - \sigma(T_1)|.$$

A median consensus tree of a profile of unrooted (rooted) trees  $\mathcal{T} = \{T_1, \dots, T_k\}$  on leaf set  $L$  is any unrooted (rooted) tree on  $L$  that minimises

$$d_s(T, \mathcal{T}) = \sum_i d_s(T, T_i).$$

The standard median tree has been characterised in terms of majority rule trees. If the number  $k$  of trees is odd then the median tree for the profile equals the majority rule tree. If the number of trees is even then the majority rule tree is still a median tree, but so is any tree containing the splits (clusters) of the majority rule tree and any splits (clusters) that appear in exactly half the trees [16].

A binary median tree is any binary tree that minimises  $d_s(T, \mathcal{T})$  over all binary trees  $T$ . In general, determining binary median trees is an NP-hard problem [79].

We show later that a median binary tree can be found in polynomial time for two trees.

The asymmetric median tree of a profile is the tree that minimises

$$\sum_i |\beta(T_i) - \beta(T)|$$

in the unrooted case, or

$$\sum_i |\sigma(T_i) - \sigma(T)|$$

in the rooted case [91].

### 6.2.9 Average Consensus Trees

So far, all of these consensus methods have only been applicable to trees without edge weights. This next method attempts to remedy this situation. Let  $\mathcal{T}$  be a profile of edge weighted trees. To each tree in  $\mathcal{T}$  there corresponds a metric  $p_i$  that is the leaf to leaf metric for  $T_i$  (Chapter Five). An **Average Consensus tree** [35] is an edge weighted tree  $T$  with leaf to leaf metric  $\delta$  that minimises

$$\sum_i \sum_{x,y \in S} (\delta(x,y) - p_i(x,y))^2.$$

This is really only a median tree with a tree to tree metric derived from the Euclidean distance between the various leaf to leaf metrics. For each  $x, y \in S$ , let  $\bar{p}(x,y)$  be the average of  $p_i(x,y)$ . Then the average consensus tree minimises

$$\sum_{x,y \in S} (\delta(x,y) - \bar{p}(x,y))^2.$$

Note that the average consensus tree method does not always produce a unique tree.

### 6.2.10 A Consensus Classification

We have established that there are a lot of consensus methods to choose from, but which should one use? What information does each method retain and how do we interpret the output trees? In a sense we face a classification problem here: we need to produce a classification of consensus methods.



A consensus method is usually introduced by comparing the amount of information it retains to the amount of information retained by the strict consensus tree. The appeal of this approach is that the strict consensus tree is almost always poorly resolved. We extend this analysis further using the partial order  $\leq$  described in Chapter Two. Recall that  $T_1 \leq T_2$  if  $T_2$  extends  $T_1$ , that is, if  $T_1$  is an induced subtree of a contraction of  $T_2$ .

Figure 40 is a classification of consensus trees. An arrow from one consensus method to another means that for all profiles of trees the consensus tree returned by the second method always extends the consensus tree, or each of the consensus trees, returned by the first method.

We also add a node  $R$  representing the set of rooted triples common to all trees. This is not necessarily a tree itself, but we include it for reference.

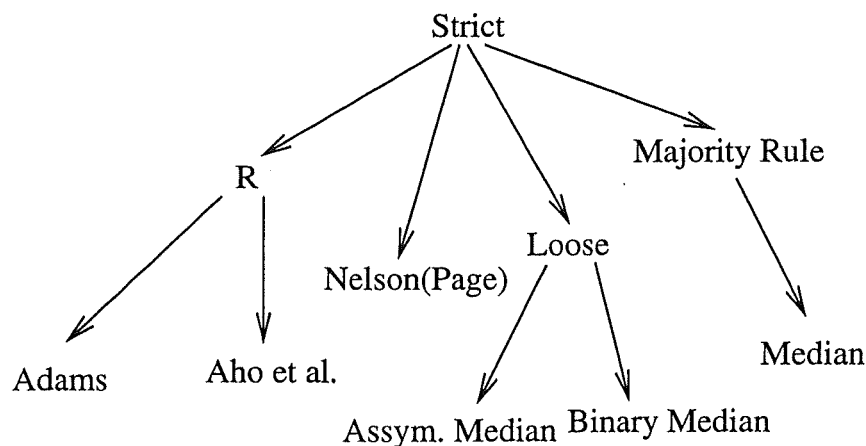


Figure 40 : A classification of consensus methods.

Several consensus methods that are different for arbitrary profiles produce the same consensus trees when applied to profiles containing only two trees. Figure 41 portrays the subtree relationships between the consensus techniques on profiles containing only two trees.

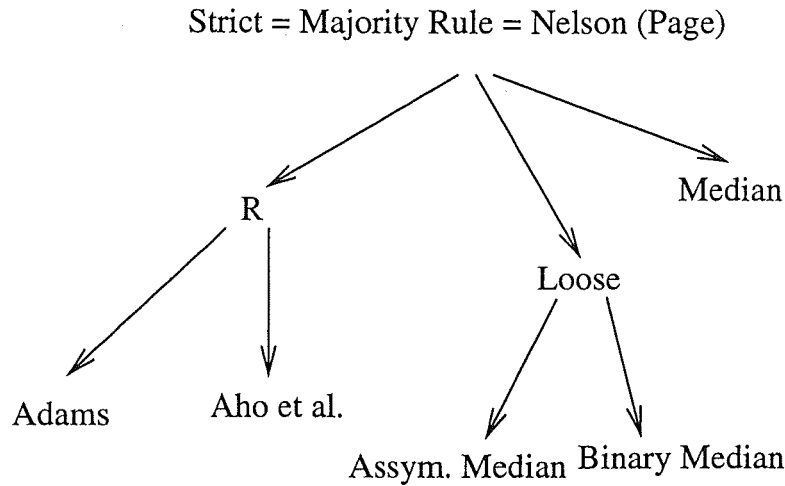


Figure 41 : A classification of consensus methods for two trees.

### 6.3 Consensus trees via axioms

McMorris [81] described two related approaches to the study of consensus methods. The first is devising new methods and then determining which mathematical properties they satisfy. The second is dreaming up a list of desirable mathematical properties and then trying to construct consensus methods that satisfy them. The previous section was based on the first approach. We now survey the second.

Much of the work on axioms for consensus trees derives from the social choice theory of Arrow [8]. Many of the following axioms are straightforward analogues to Arrows standard consensus axioms.

The consensus functions we are examining operate on sets of trees. This is not universal practice. The axiomatic approaches of [19, 18, 81, 86, 78] assume that the consensus functions are defined on ordered tuples of trees. In this context, consensus functions on sets become consensus functions satisfying the symmetry axiom [16, 77] that ignore repetition of trees within the input tuple. By looking only at consensus functions on sets of trees we have to exclude or modify several of the standard consensus axioms (see [78, 86, 14, 17, 18] for some of these).

For the following, let  $c$  be a consensus function, let  $P = \{T_1, T_2, T_3, \dots, T_k\}$  and

$P' = \{T'_1, T'_2, T'_3, \dots, T'_k\}$  be arbitrary profiles and let  $L = \mathcal{L}(T_i)$ . Let  $T - X$  denote the tree with clusters  $\sigma(T) - \{X\}$ , and let  $P - X$  denote  $\{T_1 - X, T_2 - X, \dots, T_k - X\}$ .

### General Axioms

$A_1$  *Efficiency (i)*

$$c(\{T\}) = T. \text{ [16, 14]}$$

$A_2$  *Independence (of irrelevant alternatives)*

$$\text{If } P|_X = P'|_X \text{ then } c(P)|_X = c(P')|_X. \text{ [17, 18, 19, 81, 86]}$$

$A_3$  *Independence (ii)*

$$\text{For any } X \subseteq S, c(P|_X) = c(P)|_X. \text{ [19].}$$

### Cluster Axioms (for rooted trees)

$A_4$  *Pareto*

$$\text{If } X \in \cap_i \sigma(T_i) \text{ then } X \in c(P). \text{ [17, 18, 77, 86, 78]}$$

$A_5$  *co-Pareto*

$$\sigma(c(P)) \subseteq \cup_i \sigma(T_i). \text{ [77]}$$

$A_6$  *Removal independence*

$$\text{If } P|_X - X = P'|_X - X \text{ then } c(P)|_X - X = c(P')|_X - X. \text{ [17, 18, 19]}$$

$A_7$  *Weak independence*

$$\text{If } P|_X = P'|_X \text{ then } c(P)|_X - X = c(P')|_X - X. \text{ [18]}$$

$A_8$  *Autonomy*

$$\text{Given any } X \subseteq S \text{ there is a profile } P \text{ such that } X \in \sigma(c(P)). \text{ [77]}$$

$A_9$  *Efficiency (ii)*

If every tree  $T_i$  in  $P$  either has  $X$  as its only non-trivial cluster, or equals the fan tree, then  $c(P)$  equals the fan tree or the tree with one non-trivial cluster  $X$ . [16, 14]

$A_{10}$  *Betweenness (faithfulness)*

$$\text{Given any choice of } X_i \in \sigma(T_i) \text{ for each } i, \text{ there is some cluster } B \in \sigma(c(P))$$

such that

$$\cap_{i=1}^k X_i \subseteq B \subseteq \cup_{i=1}^k X_i. [86]$$

Most of the above have obvious equivalents for unrooted trees and splits. There are also consensus axioms based on other structures in the tree, for instance, quartets and rooted triples:

#### Other structural axioms

$A_{11}$  *Pareto on quartets*

If  $ab|cd \in \cap_i q(T_i)$  then  $ab|cd \in q(c(P))$ .

$A_{12}$  *co-Pareto on quartets*

$q(c(P)) \subseteq \cup_i q(T_i)$ .

$A_{13}$  *Pareto on rooted triples*

If  $ab|c \in \cap_i r(T_i)$  then  $ab|c \in r(c(P))$ .

$A_{14}$  *co-Pareto on rooted triples*

$r(c(P)) \subseteq \cup_i r(T_i)$ .

As could be expected there are many logical dependencies between these axioms, as well as sets of axioms for which no consensus method exists.

1. If  $c$  satisfies  $A_{10}$  then it satisfies  $A_4$  [86].
2. If  $c$  satisfies  $A_2$  and  $A_4$  then it satisfies  $A_{10}$  [86].
3. If  $c$  satisfies  $A_2$  then it satisfies  $A_7$  [19].
4. If  $c$  satisfies  $A_3$  then it satisfies  $A_2$  and  $A_7$  [19].
5. If  $c$  satisfies  $A_6$  then it satisfies  $A_2$  and  $A_7$  [19].
6. There is no  $c$  satisfying  $A_2$  and  $A_{11}$  [81, 78].
7. There is no  $c$  satisfying  $A_4$  and  $A_6$  [17].

We present several new logical connections between consensus axioms, all of which follow directly from the definitions.

**Theorem 6.3** 1. If  $c$  satisfies  $A_9$  then it satisfies  $A_8$ .

2. If  $c$  satisfies  $A_1$  then it satisfies  $A_8$ .

3. If  $c$  satisfies  $A_{11}$  then it satisfies  $A_4$  (for splits).

4. If  $c$  satisfies  $A_{13}$  then it satisfies  $A_4, A_8, A_9$  and  $A_{10}$ .

5. If  $c$  satisfies  $A_5$  then it satisfies  $A_{11}$  (unrooted) or  $A_{13}$  (rooted).

## 6.4 Clique Consensus methods

In his discussion of the Nelson consensus tree Swofford [107] suggested that the construction method of Page be altered to include all clusters in the input trees, even if they are not replicated. It is surprising that no one had suggested that earlier. Indeed there is very little mention of a whole class of related consensus methods that appears to be highly intuitive.

The basic idea behind these methods, which we will call **Clique Consensus Methods**, is to take all of the clusters (or splits) of the input trees, apply a weighting based on frequency or edge weightings, and then find the maximum weight clique or cliques to give a consensus tree. In the case that there is more than one maximum weight clique one could be chosen by some other criterion or, as in Page's method [89], the intersection of these cliques could be used.

We define the **Maximum Edge Weight Consensus Tree**. This tree, like the Average Consensus Tree, takes account of edge weights in the input set. A consensus method should be biased towards including heavily weighted edges.

Given a profile  $\mathcal{T} = \{T_1, \dots, T_k\}$  of edge weighted trees construct the set  $S$  containing all of the splits (or clusters) of the original set. Weight the elements of  $S$  by the sum of the corresponding edges in the trees of  $\mathcal{T}$ . The maximum edge weight consensus tree is then constructed from a maximum weight clique of  $S$ . This method could be easily modified to take account of weights assigned to whole trees.

A consensus method is called co-Pareto [77] if the splits or clusters in the consensus tree all come from the input trees. Hence a method is a Clique Consensus Method if and only if it is co-Pareto. This includes not only the Nelson-Page consensus tree but a number of the earlier methods. Given a cluster  $X$  and a profile of

rooted trees  $\mathcal{T}$ , define  $\text{freq}(X, \mathcal{T}) = |\{T \in \mathcal{T} : X \in \sigma(T)\}|$ . When  $A|B$  is a split and  $\mathcal{T}$  is a profile of unrooted trees define  $\text{freq}(A|B, \mathcal{T}) = |\{T \in \mathcal{T} : A|B \in \beta(T)\}|$ .

- If every cluster is weighted by  $\text{freq}(X, \mathcal{T}) - k + 1/2$  then the Max Clique Consensus tree equals the strict consensus tree.
- If every cluster is weighted by  $\text{freq}(X, \mathcal{T}) - k/2$  then the Max Clique Consensus tree equals the strict Majority Rule tree.
- If every cluster is weighted by the number of trees it is compatible with, minus  $(k - 1/2)$  then the Max Clique tree equals the loose consensus tree.
- If every cluster is weighted by  $\text{freq}(X, \mathcal{T}) - k/2$  then the set of median trees equals the set of maximum weight cliques.
- The asymmetric median trees correspond to maximum weight cliques when the clusters are weighted by  $\text{freq}(X, \mathcal{T})$  [91].

### 6.4.1 Complexity

Recall that the general maximum compatible subset problem is NP-hard, so describing consensus techniques in terms of clique problems is not going to instantly provide fast efficient algorithms. However the Max Clique Consensus problem is only a specific case of the maximum compatible subset problem, because the set of clusters has to come from a collection of trees. Perhaps a bound on the number of trees in the input profile will lead to polynomial time algorithms?

This is only true to an extent. We formally state the problem, then show that it has polynomial time complexity for two trees but is NP-complete for three trees. This result is a minor extension of work in [91].

#### MAX CLIQUE CONSENSUS

INSTANCE: Collection of trees  $\mathcal{T} = \{T_1, \dots, T_k\}$ . Weight function  $w : C \rightarrow \mathbb{R}$ , where  $C = \cup_i \sigma(T_i)$ . Number  $K$ .

QUESTION: Is there a tree  $T$  such that  $\sigma(T) \subseteq C$  and  $\sum_{X \in \sigma(T)} w(X) \geq K$ ?

**Theorem 6.4** *MAX CLIQUE CONSENSUS can be solved in polynomial time if  $k \leq 2$  but is NP-complete for  $k \geq 3$ .*

Given two trees  $T_1$  and  $T_2$  construct the incompatibility graph with vertices corresponding to the clusters of  $C = \sigma(T_1) \cup \sigma(T_2)$  and an edge between two vertices if and only if the corresponding splits are incompatible. Since the clusters in each tree are compatible this graph  $G$  must be bipartite. Assign the weights to the vertices equal to weights of the corresponding splits. We want to find the independent set in  $G$  that has the maximum weight. This can be done using the “Hungarian” matching based algorithms [31], taking at most  $O(n^3)$  time.

For the second part we only have to observe that calculating the Asymmetric Median Tree for three trees is NP-complete [91], and this is only a special case of MAX CLIQUE CONSENSUS.  $\square$

### 6.4.2 Restricted Max Clique Consensus

A new consensus technique is of little use if it cannot be efficiently implemented. Therefore in its most general form the Max Clique consensus method is only of theoretical interest. However, in the light of the *Hunting for Trees* algorithms in Chapter 4 we can provide polynomial time algorithms for a restricted, but still useful, variant.

We begin with a special case and then generalise.

#### Maximum Binary Clique Consensus

INSTANCE: Profile  $\mathcal{T} = \{T_1, \dots, T_k\}$  of rooted binary trees.

PROBLEM: Find a tree  $T$  that maximises  $\sum_{X \in \sigma(T)} \text{freq}(X, \mathcal{T})$  over all binary trees with clusters in  $\cup_i \sigma(T_i)$ .

Since  $\mathcal{T}$  contains binary trees, the set  $C = \cup_i \sigma(T_i)$  will always contain enough clusters to construct a binary tree. Therefore we can always find such a tree. There are at most  $O(nk)$  clusters in  $C$ , so by using the algorithms in Chapter Four we can calculate this maximum weight binary tree in  $O(n^2k^2)$  time.

As with the *Hunting for Trees* algorithm, we do not have to restrict ourselves to binary trees, and can generalise the above consensus problem to  $d$ -trees. In particular

we obtain versions of the Maximum Weight Consensus Tree and the Asymmetric Median Tree. These are both NP-hard to construct in the general case but with the restriction of degree bound become solvable in polynomial time.

### Maximum weight consensus $d$ -tree

INSTANCE: Profile  $\mathcal{T}$  of edge-weighted trees on leaf set  $L$ , and a degree bound  $d$ .  
 PROBLEM: Find a  $d$ -tree with maximum summed edge weights, where each edge is weighted by the sum of the corresponding edge weights in trees in  $\mathcal{T}$  and each edge appears in at least one tree in  $\mathcal{T}$ , or show that no such tree exists.

A tree that is a solution to this problem is called a **maximum weight consensus  $d$ -tree**. A maximum weight consensus  $d$ -tree can be constructed, or shown not to exist, in  $O(n^2 d K^{d-1})$  time, where  $n = |L|$  and  $K = |\cup_i C(T_i)|$  [25].

### Restricted asymmetric median $d$ -tree

INSTANCE: Profile  $\mathcal{T}$  of trees on leaf set  $L$ , and a degree bound  $d$ .  
 PROBLEM: Find a tree which minimises

$$\sum_i |\beta(T_i) - \beta(T)|$$

of all  $d$ -trees that have splits from trees in  $\mathcal{T}$ , or show that no such tree exists.

A tree that solves this problem is called a **restricted asymmetric median  $d$ -tree** and can also be constructed in  $O(n^2 d K^{d-1})$  time, where  $K = |\cup_i C(T_i)|$  and  $n = |L|$  [25].

In general we cannot guarantee the existence of a restricted median  $d$ -tree or a maximum weight consensus  $d$ -tree. However, *if the input profile contains at least one  $d$ -tree then their existence is guaranteed*. For example if the input profile contains at least one binary tree, as is often the case, then for all  $d \geq 3$  there exist restricted median  $d$ -trees, asymmetric median  $d$ -trees and, if the trees in the input profile have weighted edges, maximum weight consensus  $d$ -trees.



## 6.5 Agreement Methods

All of the consensus methods we have described take a collection of trees and return a tree, or set of trees, *on the same leaf set*. However this might not be the best way to represent the information shared by the trees. For example, consider the two trees (1) and (2) in Figure 42.

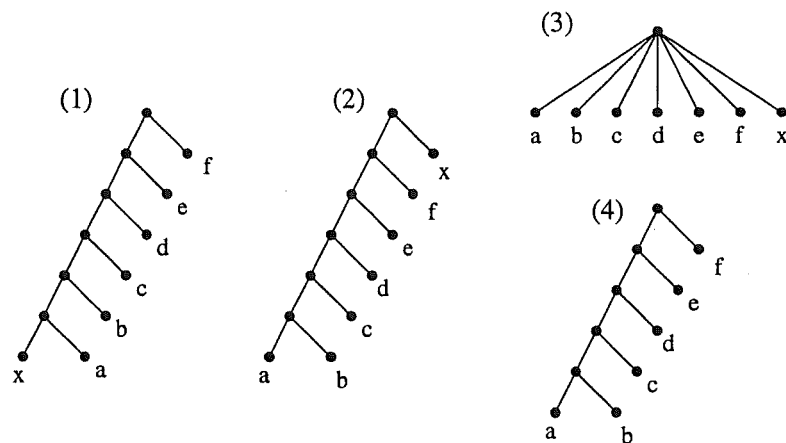


Figure 42 : The strict consensus tree of (1) and (2) equals the fan tree (3), but the MAST tree contains a lot more information (4).

Trees (1) and (2) are identical except for the position of the taxa  $x$ , and yet the strict consensus tree of these trees contains no information (3). A far more representative tree (4) is obtained by removing the taxa  $x$ .

We describe several methods for extracting information common to trees that do not return trees with a full leaf set. Collectively these methods are called **Agreement Methods**.

### 6.5.1 Reduced Consensus Profiles

One problem with many consensus techniques is that the consensus tree can contain information that is not present in all of the input trees, and sometimes not in any of the trees. It has been long recognised that the Adams consensus tree suffers from

this problem [107] so that interpretation of the tree must be made with considerable care. Wilkinson [116] proposes to reduce ambiguity by outputting not the single Adams tree, but the set of trees  $T$  satisfying:

1.  $T$  is an induced subtree of the Adams tree.
2.  $T \trianglelefteq T_i$  for all trees  $T_i$  in the input profile.
3. There is no tree  $T' \neq T$  satisfying 1 and 2 such that  $T \trianglelefteq T'$ .
4.  $T$  is not a fan tree.

This possibly empty collection of trees is called the Reduced Adams Consensus (RAC) profile for  $\mathcal{T}$ . Given a profile  $\mathcal{T} = \{T_1, \dots, T_k\}$ , let  $R = \bigcap_i r(T_i)$  and let  $T^{AD}$  be the Adams consensus tree for  $\mathcal{T}$ . The RAC profile is equal to the maximal elements of the set

$$\{T|_X^{AD} : \emptyset \subset r(T|_X^{AD}) \subseteq R\}.$$

In addition to the RAC profile, Wilkinson describes the Reduced Cladistic Consensus profile. Given a profile of trees  $\mathcal{T}$ , the RCC profile contains those trees  $T$  satisfying

1.  $T \trianglelefteq T_i$  for all trees  $T_i \in \mathcal{T}$ .
2. There is no tree  $T' \neq T$  satisfying 1 such that  $T \trianglelefteq T'$ .
3.  $T$  is not a fan tree.

In terms of our earlier notation, the RCC profile equals the maximal members of the set

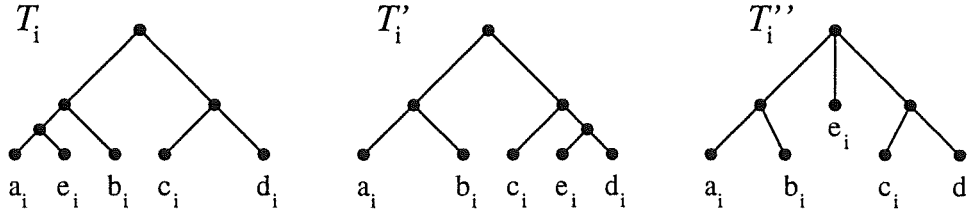
$$\{T : \emptyset \subset r(T) \subseteq R\}.$$

These approaches may work well for trees with a small number of taxa, but are not really practical for large trees. Wilkinson does not appear to have considered the problem that the size of the RAC and RCC profiles can grow exponentially with respect to the number of taxa.

**Theorem 6.5** *For each  $n > 1$  there exist trees  $T$  and  $T'$  with  $5n$  leaves such that the RAC and RCC profiles for  $T$  and  $T'$  contain more than  $4^n$  trees.*

*Proof*

For each  $i = 1, \dots, n$  construct  $T_i, T'_i, T''_i$  as in Figure 43.

Figure 43 : The trees  $T_i, T'_i, T''_i$  in the proof of Theorem 6.5.

Note that  $T''_i$  is the Adams consensus tree of  $T_i$  and  $T'_i$ .

Construct a rooted caterpillar tree with leaves  $x_1, x_2, \dots, x_n$ , labelled in any order. For  $T$  replace each leaf  $x_i$  of the caterpillar tree with the tree  $T_i$ . For  $T'$  replace each leaf  $x_i$  of the caterpillar tree with  $T'_i$ . The Adams consensus tree of  $T$  and  $T'$ , which we will denote  $T''$  is the caterpillar tree with each leaf  $x_i$  replaced by  $T''_i$ .

For each  $i$ , choose  $u_i \in \{a_i, b_i\}$  and  $v_i \in \{c_i, d_i\}$ . Define

$$S = \bigcup_{i=1}^n \{e_i, u_i, v_i\}.$$

Then  $T''_S \trianglelefteq T$  and  $T''_S \trianglelefteq T'$ . This will not be true if we add any more leaves to  $S$ , so  $T''_S$  is in the RAC profile for  $T$  and  $T'$ . The tree  $T''_S$  is also in the RCC profile for  $T$  and  $T'$  because we cannot add any edges to  $T''_S$  and still get  $T''_S \trianglelefteq T$  and  $T''_S \trianglelefteq T'$ .

For each  $i = 1, \dots, n$  there are four different choices of  $u_i$  and  $v_i$ . Hence there are at least  $4^n$  trees in the RAC and RCC profiles of  $T$  and  $T'$ .  $\square$

It appears that a further criterion needs to be included in the definition of the RAC and RCC profiles, in order to avoid this mushrooming effect. For example we could check each candidate for the RAC or RCC profile as soon as we construct it, making sure that it contains at least one rooted triple not contained in the previous trees. In this way the RAC or RCC profile would not be unique, but its size would be limited to  $O(n^3)$  trees,  $n$  being the number of leaves.

### 6.5.2 Agreement subtrees

A tree  $T$  with leaf set  $X$  is an **agreement subtree** of a profile of trees  $\mathcal{T} = \{T_1, \dots, T_k\}$  if

$$T = T_{1|X} = T_{2|X} = \dots = T_{k|X},$$

that is, if  $T$  is an induced subtree of all the trees  $T_i$ . Agreement subtrees provide an intuitive way to represent branching information shared by a collection of trees. They were introduced by Finden and Gordon [53] who called them ‘common pruned trees’.

A **Maximum Agreement Subtree** or MAST of a profile is an agreement subtree with the maximum number of leaves. There can be several maximum agreement subtrees for a given profile, indeed exponentially many [72]. An exact, but super-polynomial time, algorithm for computing a MAST of two binary trees was presented by Kubicka *et al.* [71] (submitted in 1992). Polynomial time algorithms for computing a MAST of two trees were developed independently by Steel and Warnow [103] and Goddard *et al.* [57]. The problem of computing MAST for three or more trees was shown to be NP-hard by Amir and Keselman [7]. Even approximating the size of the MAST tree is difficult [63].

Fortunately in real life the trees that people want to calculate MAST trees for tend to have small vertex degree. If we can place a degree bound on one of the input trees then a Maximum Agreement Subtree can be constructed in polynomial time. Amir and Keselman [7] discovered an  $O(knd^{d+1} + n^{2d})$  algorithm. Farach *et al.* [50] improved this bound, describing an  $O(kn^3 + n^d)$  algorithm.

The algorithm for MAST that we give here also has complexity  $O(kn^3 + n^d)$  but is simpler than the algorithm of [50] thanks to the use of rooted triples and closed sets. Let  $R$  be the set of rooted triples common to the trees in a profile  $\mathcal{T}$ . If one of the trees in  $\mathcal{T}$  is binary then the number of leaves in the MAST tree of  $\mathcal{T}$  is just the maximum over all leaves  $a, b$  of

$$MAST(a, b) = \max\{MAST(a, x) + MAST(b, y) : x \in A, y \in B\}$$

where  $A = \{x : ax|b \in R\} \cup \{a\}$  and  $B = \{y : by|a \in R\} \cup \{b\}$ .

We extend our algorithm to solve related problems: constructing an agreement subtree with a maximum number of edges, constructing an agreement subtree compatible with the maximum number of rooted triples and constructing an agreement

subtree compatible with the maximum number of quartets.

First we present an algorithm for constructing all agreement subtrees. The algorithm provides the basis of all of the agreement subtree algorithms.

In all of the following, let  $\mathcal{T} = \{T_1, \dots, T_k\}$  be a profile of rooted trees on leaf set  $L$  such that at least one of the trees in  $\mathcal{T}$  is a  $d$ -tree. Given two leaves  $a, b \in L$  let  $\mathcal{T}(a, b)$  denote the set of agreement subtrees  $T$  of  $\mathcal{T}$  such that  $a$  and  $b$  are in different maximal subtrees of  $T$ , that is, the root of  $T$  equals the least common ancestor of  $a$  and  $b$ . We define  $\mathcal{T}(a, a)$  to be the singleton vertex  $a$ . Finally, let

$$R = \bigcap_i r(T_i) \text{ and } F = \bigcap_i f(T_i).$$

Refer to Chapter 1 for definitions of  $r(T)$  and  $f(T)$ .

The basis for our agreement subtree techniques is the following observation:

**Lemma 6.6** *A tree  $T$  is an agreement subtree for  $\mathcal{T}$  if and only if*

$$r(T) \subseteq R \text{ and } f(T) \subseteq F.$$

*Proof*

$T$  is an agreement subtree for  $\mathcal{T}$  if and only if  $T$  is an induced subtree of each tree  $T_i \in \mathcal{T}$  if and only if  $r(T) \subset r(T_i)$  and  $f(T) \subset f(T_i)$  for all  $T_i \in \mathcal{T}$  if and only if

$$r(T) \subseteq R \text{ and } f(T) \subseteq F. \quad \square$$

Though we have so far only discussed inference rules for rooted triples and quartets there are some extremely useful inference rules involving fan trees.

**Lemma 6.7** (i) *If  $wx|y, yz|w \in R$  then  $wx|z \in R$ .*

(ii) *If  $wx|y \in R$  and  $(w, y, z) \in F$  then  $wx|z \in R$ .*

(iii) *Suppose that  $xy|u \notin R$  for all  $u$  in some set  $U \subset L$ . If  $xy|z \in R$  then  $u_1u_2|z \in R$  for all  $u_1, u_2 \in U \cup \{x, y\}$ . If  $xy|z \in R$  and  $(w, x, z) \in F$  then  $(w, z, u) \in F$  for all  $u \in U$  and  $u_1u_2|w \in R$  for all  $u_1, u_2 \in U \cup \{x, y\}$ .*

*Proof*

Part (i) is just the standard dyadic rule for rooted triples (Chapter 3, section 3.2).

Part (ii) follows from the fact that every tree that extends  $wx|y$  and  $(x, y, z)$  must also extend tree (1) in Figure 44. For (iii) we note that if  $xy|z \in r(T)$  for some tree

$T$  and  $xy|u \notin r(T)$  then there are only three possibilities for the subtree induced by  $\{x, y, z, u\}$  (Figure 44 (2)). The second part of (iii) is similar.  $\square$

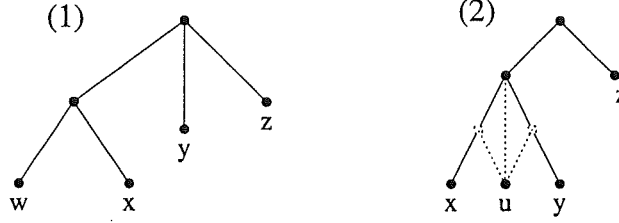


Figure 44 : The two trees (1) and (2) for the proof of Lemma 6.7.

The procedure AGSUBTREE (Algorithm 20) returns an arbitrary member of  $\mathcal{T}(a, b)$ .

**Theorem 6.8** (i) The algorithm AGSUBTREE always returns a tree in  $\mathcal{T}(a, b)$ .  
(ii) The algorithm AGSUBTREE can return any tree in  $\mathcal{T}(a, b)$ .

*Proof*

(i) We prove the theorem using induction with respect to the partial order  $\prec$  defined on pairs of leaves in  $T_1$  (see section 2.3.5, page 24). The theorem is clearly true if  $a = b$ . Suppose that it is true for all  $\{c, d\} \prec \{a, b\}$ .

Let  $T$  be a tree returned by AGSUBTREE( $a, b$ ). Let  $S = \{s_1, s_2, \dots, s_m\}$  be the clique of  $G$  chosen in step 7. We have  $(a, s_i, b) \in F$  for all  $s_i \in S$  and  $(a, s_i, s_j) \in F$  for all  $s_i, s_j \in S$ . By Lemma 6.7 (i) we have  $(s_i, s_j, b) \in F$  for all  $s_i, s_j \in S$ . Therefore all of the fans with leaves in  $S \cup \{a, b\}$  are in  $F$  and so the fan tree  $T_F$  of Figure 45 is an agreement subtree of  $\mathcal{T}$ .

Now by the construction of  $A$  and  $B$  in line 2 of AGSUBTREE, if  $a \neq x^*$  then  $ax^*|b \in R$  and if  $b \neq y^*$  then  $by^*|a \in R$ . By repeated application of Lemma 6.7 (ii) we see that the tree  $T_F^*$  of Figure 46 is also an agreement subtree for  $\mathcal{T}$ .

Note that  $a$  could equal  $x^*$  or  $b$  could equal  $y^*$ , in which case the pendant pairs would be replaced by a single vertex. The resulting tree would still be an agreement subtree.

Procedure AGSUBTREE( $a, b$ )

1. IF  $a = b$  THEN RETURN the vertex  $a$ .
  2. Construct:
 
$$A \leftarrow \{x : ax|b \in R\} \cup \{a\}$$

$$B \leftarrow \{y : by|a \in R\} \cup \{b\}$$

$$C \leftarrow \{z : (azb) \in F\}$$
  3. Choose  $x^* \in A$ .
  4. Choose  $y^* \in B$ .
  5. For each  $z \in C$ , choose  $z^* \in C$  such that either  $z^* = z$  or  $z^*z|a \in R$ .
  6. Construct a graph  $G$  with vertex set  $C$  and an edge between  $v$  and  $w$  if and only if  $(avw) \in F$ .
  7. Choose a clique  $S$  of  $G$ .
  8. Construct the trees AGSUBTREE( $a, x^*$ ), AGSUBTREE( $b, y^*$ ) and AGSUBTREE( $s, s^*$ ) for all  $s \in S$ . Attach the roots of these trees to a new root  $v$ .
  9. RETURN this tree.
- END.

Algorithm 20 : AGSUBTREE

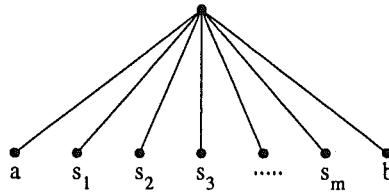
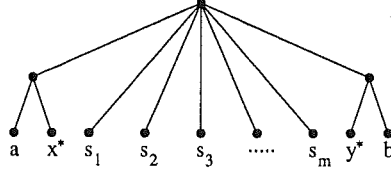
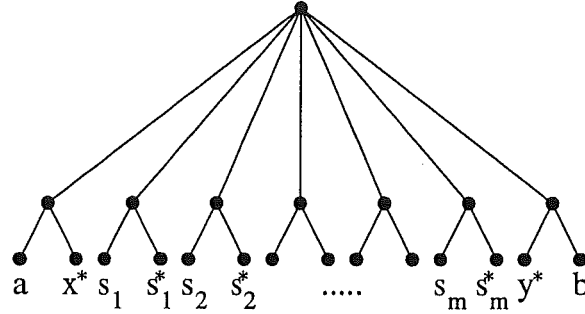


Figure 45 : Tree  $T_F$  in the proof of Theorem 6.8.

Figure 46 : Tree  $T_F^*$  in the proof of Theorem 6.8.

For each  $i = 1, 2, \dots, m$  such that  $s_i \neq s_i^*$  we have from lines 2 and 5 that  $s_i s_i^* | a \in R$  and  $(a, s_i^*, b) \in F$ . Again using Lemma 6.7 together with a smattering of Lemma 6.6 we conclude that the tree  $T_\alpha$  of Figure 47 is an agreement subtree for  $\mathcal{T}$ .

Figure 47 : Tree  $T_\alpha$  in the proof of Theorem 6.8.

As before, the pendant pairs  $s_i, s_i^*$  would be replaced by a single leaf if  $s_i = s_i^*$ .

Since  $T_\alpha$  is an agreement subtree for  $\mathcal{T}$  it is an induced subtree of  $T_1$ , from which we can conclude that  $\{a, x^*\} \prec \{a, b\}$  in  $T_1$  and  $\{b, y^*\} \prec \{a, b\}$  in  $T_1$  and  $\{s_i, s_i^*\} \prec \{a, b\}$  in  $T_1$  for all  $i = 1, 2, \dots, m$ . Let  $S_a, S_b$  and  $S_1, \dots, S_m$  be the trees returned by the recursive calls in line 8:  $\text{AGSUBTREE}(a, x^*)$ ,  $\text{AGSUBTREE}(b, y^*)$  and  $\text{AGSUBTREE}(s_i, s_i^*)$  for  $i = 1, \dots, m$ . By the induction step these are all agreement subtrees for  $\mathcal{T}$ , so  $r(S_i) \subseteq R$  and  $f(S_i) \subseteq F$  for all  $i \in \{1, \dots, m, a, b\}$ .



Given any  $\{i, j, k\} \subseteq \{1, \dots, m, a, b\}$ , Lemma 6.7 (iii) tells us that  $(u_i, u_j, u_k) \in F$  for all  $u_i \in \mathcal{L}(S_i)$ ,  $u_j \in \mathcal{L}(S_j)$  and  $u_k \in \mathcal{L}(S_k)$ . Lemma 6.7 (iii) also gives us  $u_i v_i | w_j \in R$  for all  $u_i, v_i \in \mathcal{L}(S_i)$ ,  $w_j \in \mathcal{L}(S_j)$  and  $\{i, j\} \subseteq \{1, \dots, m, a, b\}$ . Put

$$R_P = \{u_1 u_2 | v : u_1, u_2 \in \mathcal{L}(S_j), v \in \mathcal{L}(S_k), \{j, k\} \subseteq \{1, \dots, m, a, b\}\}$$

and

$$F_P = \{(u, v, w) : u \in S_j, v \in S_k, w \in S_l, \{j, k, l\} \subseteq \{1, \dots, m, a, b\}\}.$$

What we have shown is that  $R_P \subseteq R$  and  $F_P \subseteq F$ .

Since

$$r(T) = r(S_0) \cup r(S_1) \cup r(S_2) \cup \dots \cup r(S_{m+1}) \cup R_P$$

and

$$f(T) = f(S_0) \cup f(S_1) \cup f(S_2) \cup \dots \cup f(S_{m+1}) \cup F_P$$

we have that  $r(T) \subseteq R$  and  $f(T) \subseteq F$  so  $T$  is an agreement subtree for  $\mathcal{T}$ , completing part (i) of the proof.

(ii) Suppose that  $T$  is member of  $\mathcal{T}(a, b)$ . Let  $S_a$  be the maximal subtree containing  $a$ , let  $S_b$  be the maximal subtree containing  $b$ , and let  $S_1, S_2, \dots, S_m$  be the remaining maximal subtrees, if there are any. Choose  $x^*$  in  $\mathcal{L}(S_a)$  such that the lowest common ancestor of  $a$  and  $x^*$  equals the root of  $S_a$  and choose  $y^*$  such the lowest common ancestor of  $b$  and  $y^*$  equals the root of  $S_b$ . For each subtree  $S_i$  choose two leaves  $s_i$  and  $s_i^*$  so that the lowest common ancestor of  $s_i$  and  $s_i^*$  equals the root of  $S_i$ . It is not hard to see that  $x^*, y^*$  and each  $s_i$  and  $s_i^*$  could be selected by the algorithm. By the induction step, the trees  $S_a, S_b$  and  $S_i, i = 1, 2, \dots, m$  could be returned by the algorithm. Hence the algorithm  $\text{AGSUBTREE}(a, b)$  can choose any agreement subtree  $T \in \mathcal{T}(a, b)$ .  $\square$

### 6.5.3 Maximum Agreement Subtree (MAST)

The algorithm  $\text{AGSUBTREE}$  provides the means to optimise agreement subtrees using dynamical programming. Given the appropriate function, we can find the maximum on subtrees, then build up until we have found the maximum over the whole

domain. The first example of this approach is the classical Maximum Agreement Subtree problem - find an agreement subtree with the largest number of leaves.

Algorithm 21 returns the number of leaves in the MAST and can be easily modified to return the actual tree after this value is calculated.

Procedure MAST( $a, b$ )

1. IF  $a = b$  THEN RETURN 1.

2. Construct:

$$A \leftarrow \{x : ax|b \in R\} \cup \{a\}$$

$$B \leftarrow \{y : by|a \in R\} \cup \{b\}$$

$$C \leftarrow \{z : (azb) \in F\}$$

3. Choose  $x^* \in A$  that maximises MAST( $a, x^*$ )

4. Choose  $y^* \in B$  that maximises MAST( $b, y^*$ )

5. For each  $z \in C$ , choose  $z^*$  that maximises MAST( $z, z^*$ ) over all  $\{z' \in C : zz'|a \in R\} \cup \{z\}$ .

6. Construct a weighted graph  $G$  with vertex set  $C$  and an edge between  $v$  and  $w$  if and only if  $(avw) \in F$ . Weight each vertex  $v$  of  $G$  by MAST( $v, v^*$ ).

7. Choose a maximum weight clique  $S$  of  $G$

8. RETURN MAST( $a, x^*$ ) + MAST( $b, y^*$ ) +  $\sum_{s \in S} \text{MAST}(s, s^*)$

END.

Algorithm 21 : MAST

**Theorem 6.9** *The procedure MAST( $a, b$ ) returns the largest value of  $|\mathcal{L}(T)|$  over all  $T \in \mathcal{T}(a, b)$ .*

*Proof*

The algorithm works correctly when  $a = b$ . Assume that it works for all  $\{c, d\} \prec \{a, b\}$  in  $T_1$  (see section 2.3.5, page 24 for the definition of  $\prec$ ).

Choose  $T \in \mathcal{T}(a, b)$ . Let  $S_a$  be the maximal subtree of  $T$  containing  $a$ , let  $S_b$  be the maximal subtree containing  $b$ , and let  $S_1, S_2, \dots, S_m$  be the remaining

maximal subtrees, if there are any. Choose  $x^*, y^*$ , the leaves  $s_i$  and the leaves  $s'_i$ , so that  $S_a \in \mathcal{T}(a, x^*)$ ,  $S_b \in \mathcal{T}(b, y^*)$  and  $S_i \in \mathcal{T}(s_i, s'_i)$  for  $i = 1, \dots, m$ . The set  $\{s_1, \dots, s_m\}$  forms a clique in the graph  $G$  (step 6).

Using the induction step we have:

$$\begin{aligned}
 |\mathcal{L}(T)| &= |\mathcal{L}(S_a)| + |\mathcal{L}(S_b)| + |\mathcal{L}(S_1)| + \dots + |\mathcal{L}(S_m)| \\
 &\leq \text{MAST}(a, x^*) + \text{MAST}(b, y^*) + \sum_i \text{MAST}(s_i, s'_i) \\
 &\leq \max_{x \in A} \text{MAST}(a, x) + \max_{y \in B} \text{MAST}(b, y) + \text{Maximum clique weight in } G. \\
 &= \text{value returned by MAST}(a, b)
 \end{aligned}$$

Conversely, suppose that the leaves  $x^*, y^*$  and  $\{s_i, s'_i\}$  were chosen as in the algorithm  $\text{MAST}(a, b)$ . By the induction step there are trees  $S_a \in \mathcal{T}(a, x^*)$ ,  $S_b \in \mathcal{T}(b, y^*)$  and  $S_i \in \mathcal{T}(s_i, s'_i)$  for  $i = 1, 2, \dots, m$  such that  $|\mathcal{L}(S_a)| = \text{MAST}(a, x^*)$ ,  $|\mathcal{L}(S_b)| = \text{MAST}(b, y^*)$ , and  $|\mathcal{L}(S_i)| = \text{MAST}(s_i, s'_i)$  for  $i = 1, 2, \dots, m$ .

$i = 1, 2, \dots, m$ . Create a new vertex and connect it to the roots of these trees. The number of leaves in this tree will then equal the value returned by  $\text{MAST}(a, b)$ . By Theorem 6.8 this tree is an agreement subtree of  $\mathcal{T}$  and so is in  $\mathcal{T}(a, b)$ . It must therefore be a tree in  $\mathcal{T}(a, b)$  with the largest number of leaves.  $\square$

**Theorem 6.10** *We can use the procedure MAST (Algorithm 21) to calculate the number of leaves in the Maximum Agreement Subtree for  $\mathcal{T} = \{T_1, T_2, \dots, T_k\}$  in  $O(kn^3 + n^d)$  time, where  $d$  is the vertex degree of at least one of the trees.*

*Proof*

Consider first the number of calculations required for one pass through the procedure MAST. The sets in step 2 can be constructed in  $O(n)$  time, which is the same time complexity as the maximisation in steps 3, 4 and 5. The step that takes the most time is finding the maximum weight clique in step 7. This takes  $O(n^{d-2})$  time. Hence for each  $a, b$  the procedure  $\text{MAST}(a, b)$  takes  $O(\max(n^{d-2}, n))$  time.

This procedure  $\text{MAST}(a, b)$  is called for every pair of leaves  $(a, b)$  in order to calculate the Maximum Agreement Subtree. Note that constructing  $R$  and  $F$  takes  $O(kn^3)$ . Hence the total time complexity is  $O(kn^3 + n^d)$  which, incidentally, is the

same time complexity as the algorithm of Farach *et al.* [50].  $\square$

If  $\mathcal{T}$  contains a binary tree then  $d = 2$  and, at each pass through the procedure there is no need to optimise over cliques. Hence we can simplify the algorithm to obtain the formula

$$\text{MAST}(a, b) = \max\{\text{MAST}(a, x) + \text{MAST}(b, y) : x \in A, y \in B\}$$

where  $A = \{x : ax|b \in R\} \cup \{a\}$  and  $B = \{y : by|a \in R\} \cup \{b\}$ .

#### 6.5.4 Maximum Information Agreement Subtree (MIST)

Swofford [107] identified a problem with the Maximum Agreement Subtree: sometimes the agreement subtree with the most leaves is less informative than an agreement subtree with fewer leaves. We reproduce his example in Figure 48.

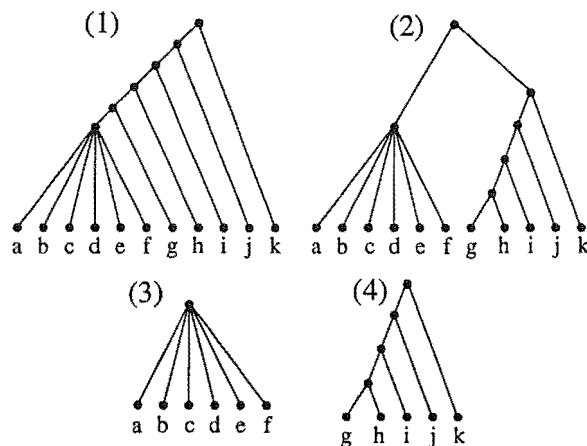


Figure 48 : An example from [107] demonstrating that the agreement subtree with the *largest* number of leaves is not always preferable.

Tree (3) is the Maximal Agreement Subtree of (1) and (2) but it conveys much less information than does a smaller agreement subtree (4).

In response to Swofford's criticism we present a new criterion for selecting agreement subtrees. In Figure 48 the preferred agreement tree is just the tree with the most non-trivial clusters. With this in mind we define the Maximum Information

Agreement Subtree (MIST) problem.

INSTANCE: Profile  $\mathcal{T} = \{T_1, \dots, T_k\}$  on leaf set  $L$ .

PROBLEM: Find an agreement subtree  $T$  for  $\mathcal{T}$  that has a maximum number of non-trivial clusters.

We show that this problem can be solved in time  $O(kn^3 + kn^d)$  where  $d$  is a degree bound on one of the trees. We actually solve a more general problem, one that combines both MAST and MIST.

INSTANCE: Profile  $\mathcal{T} = \{T_1, \dots, T_k\}$  on leaf set  $L$ . Real numbers  $\alpha, \beta \geq 0$ .

PROBLEM: Find an agreement subtree  $T$  for  $\mathcal{T}$  that maximises  $f_{\alpha\beta}(T)$ , where

$$f_{\alpha\beta}(T) = \alpha|\mathcal{L}(T)| + \beta|\{X \in \sigma(T) : X \text{ non-trivial}\}|.$$

Procedure MIST( $a, b, \alpha, \beta$ )

1. IF  $a = b$  THEN RETURN  $\alpha$ .
2. Otherwise, construct:

$$A \leftarrow \{x : ax|b \in R\} \cup \{a\}$$

$$B \leftarrow \{y : by|a \in R\} \cup \{b\}$$

$$C \leftarrow \{z : (azb) \in F\}$$

3. Choose  $x^* \in A$  that maximises MIST( $a, x^*, \alpha, \beta$ )
4. Choose  $y^* \in B$  that maximises MIST( $b, y^*, \alpha, \beta$ )
5. For each  $z \in C$ , choose  $z^*$  that maximises MIST( $z, z^*, \alpha, \beta$ ) over all  $\{z' \in C : zz'|a \in R\} \cup \{z\}$ .
6. Construct a weighted graph  $G$  with vertex set  $C$  and an edge between  $v$  and  $w$  if and only if  $(avw) \in F$ . Weight each vertex  $v$  of  $G$  by MIST( $v, v^*$ ).
7. Choose a maximum weight clique  $S$  of  $G$
8. RETURN  
 $\text{MIST}(a, x^*, \alpha, \beta) + \text{MIST}(b, y^*, \alpha, \beta) + \sum_{s \in S} \text{MIST}(s, s^*, \alpha, \beta) + \beta$   
end.

Algorithm 22 : MIST

Both of the MIST problems have unrooted equivalents. We can convert each unrooted problem into a rooted problem by solving  $n$  instances of the rooted problem, rooting the trees at a different leaf each time.

The correctness and complexity proofs are omitted because they are almost identical to those for MAST in the previous section.

**Theorem 6.11** *The procedure  $\text{MIST}(a, b, \alpha, \beta)$  returns the largest value of  $\alpha|\mathcal{L}(T)| + \beta|\{X \in \sigma(T) : |X| > 1\}|$  over all  $T \in \mathcal{T}(a, b)$ .*

**Theorem 6.12** *Using the algorithm MIST the size of the Maximum Information Subtree for  $\mathcal{T} = \{T_1, T_2, \dots, T_k\}$  can be calculated in time  $O(kn^3 + n^d)$ , where  $d$  is the vertex degree of at least one of the trees.*

### 6.5.5 Maximum Number of Triples Agreement Subtree (MAN-TAT)

Even the agreement tree with the largest number of internal edges is not always the most informative agreement tree. The example in Figure 49 also comes from Swofford [107], though not in the same section.

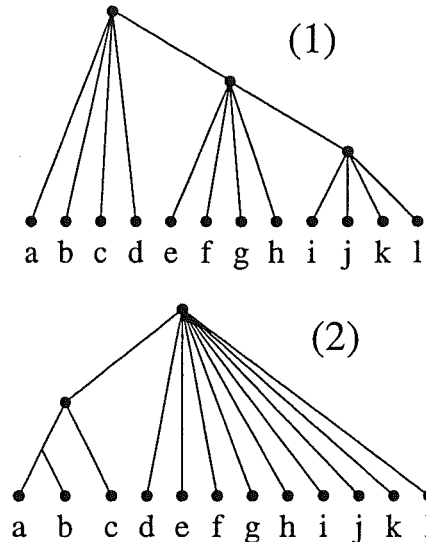


Figure 49 : An example reproduced from Swofford [107] showing that trees with the same number of non-trivial clusters convey different amounts of information.

Here we have two trees with the same number of leaves and the same number of clusters, yet the first tree (1) allows more statements of relationship to be made than the second tree. Therefore it will sometimes be necessary to optimise another with respect to another criterion, in this case the number of rooted triples in the agreement tree. In the example, tree (1) contains 136 rooted triples while tree (2) contains only 28. We define, then, the Maximum Number of Triples Agreement Subtree (MANTAT) problem.

INSTANCE: Profile  $\mathcal{T} = \{T_1, \dots, T_k\}$  on leaf set  $L$ .

PROBLEM: Find an agreement subtree  $T$  for  $\mathcal{T}$  that contains the maximum number of rooted triples.

We show that this problem can also be solved in time  $O(kn^3 + n^{2d})$ , where  $d$  is a degree bound on one of the trees. More efficient algorithms should be possible; the aim here is to show that the problem can be solved in polynomial time.

For each pair of leaves  $a, b$  the following algorithm returns an  $n$ -tuple. The  $m$ -th number in this tuple equals the maximum number of rooted triples over all trees in  $\mathcal{T}(a, b)$  with  $m$  leaves, or  $-1$  if there are no such trees. For simplicity let  $\text{MANTAT}(a, b)[m]$  denote the  $m$ -th number in the tuple returned by  $\text{MANTAT}(a, b)$ .

The function  $\rho(m_1, \dots, m_q)$  in step 18 is defined by

$$\rho(m_1, \dots, m_q) = \sum_{i \neq j} m_i(m_i - 1)m_j / 2$$

**Theorem 6.13** *The  $m$ -th entry of the vector returned by  $\text{MANTAT}(a, b)$  is equal to*

$$\max\{|r(T)| : T \in \mathcal{T}(a, b), |\mathcal{L}(T)| = m\}.$$

*Proof*

The algorithm works correctly when  $a = b$ . Assume that it works for all  $\{c, d\} \prec \{a, b\}$  in  $T_1$  (see section 2.3.5, page 24 for the definition of  $\prec$ ).

Choose  $T \in \mathcal{T}(a, b)$  and put  $m = |\mathcal{L}(T)|$ . Let  $S_a$  be the maximal subtree of  $T$  containing  $a$ , let  $S_b$  be the maximal subtree containing  $b$ , and let  $S_1, S_2, \dots, S_q$  be the remaining maximal subtrees (if any). Put  $m_a = |\mathcal{L}(S_a)|$ ,  $m_b = |\mathcal{L}(S_b)|$  and

Procedure MANTAT( $a, b$ )

1. For all  $i = 1, \dots, n$  put  $M[i] \leftarrow -1$ .
2. IF  $a = b$  THEN
3.      $M[1] \leftarrow 0$
4.     RETURN  $M$ .
5. ELSE
6.     Construct:
 
$$A \leftarrow \{x : ax|b \in R\} \cup \{a\}$$

$$B \leftarrow \{y : by|a \in R\} \cup \{b\}$$

$$C \leftarrow \{z : (azb) \in F\}$$
7.     FOR each  $m = 1, 2, \dots, n$  DO
8.         Let  $MZ[a, m]$  be the largest value of MANTAT( $a, x$ )[ $m$ ] over all  $x \in A$
9.         Let  $MZ[b, m]$  be the largest value of MANTAT( $b, y$ )[ $m$ ] over all  $y \in B$ .
10.        FOR each  $z \in C$  DO
11.           Let  $MZ[z, m]$  be the largest value of MANTAT( $z, z'$ )[ $m$ ] over all  $z' \in C$  such that  $zz'|a \in R$  or  $z' = z$ .
12.        END(FOR)
13.     END(FOR)
14.     Construct a graph  $G$  with vertex set  $C$  and an edge between  $v$  and  $w$  if and only if  $(avw) \in F$ .
15.     FOR all cliques  $S = \{s_1, \dots, s_q\}$  of  $G$  DO
16.         FOR all  $1 \leq m_a, m_1, m_2, \dots, m_q, m_b$  with sum at most  $n$  DO
17.             IF  $MZ[s_i, m_i] \geq 0$  for all  $i = a, 1, \dots, q, b$  THEN
18.                  $X \leftarrow MZ[a, m_a] + MZ[b, m_b] + \sum_{i=1}^q MZ[s_i, m_i] + \rho(m_a, m_1, \dots, m_q, m_b)$
19.                 IF  $X > M[m_a + \sum_{i=1}^q m_i + m_b]$  THEN  $M[m_a + \sum_{i=1}^q m_i + m_b] \leftarrow X$
20.             END(IF)
21.         END(FOR)
22.     END(FOR)
23.     RETURN  $M$
24. END(IF-ELSE)
- END.

Algorithm 23 : MANTAT



$m_i = |\mathcal{L}(S_i)|$  for  $i = 1, 2, \dots, q$ . We can select leaves  $x, y, s_1, \dots, s_q$  and  $s'_1, \dots, s'_q$  such that  $S_a \in \mathcal{T}(a, x)$ ,  $S_b \in \mathcal{T}(b, y)$ , and  $S_i \in \mathcal{T}(s_i, s'_i)$  for all  $i = 1, \dots, q$ . Because  $T$  is an agreement subtree  $r(T) \subseteq R$  and  $f(T) \subseteq F$  and therefore  $(a, s_i, s_j) \in F$  for all  $i, j \in \{1, \dots, q\}$ . Hence  $\{s_1, \dots, s_q\}$  forms a clique in  $G$ .

For each  $i$ ,

$$\begin{aligned} |r(S_i)| &\leq \max\{|r(T_i)| : T_i \in \mathcal{T}(s_i, s'_i), |\mathcal{L}(T_i)| = m_i\} \\ &= \text{MANTAT}(s_i, s'_i)[m_i] \\ &\leq \max\{\text{MANTAT}(s_i, \alpha)[m_i] : s_i \alpha a \in R\} \\ &= \text{MZ}[s_i, m_i] \end{aligned}$$

In the same way,  $|r(S_a)| \leq \text{MZ}[a, m_a]$  and  $|r(S_b)| \leq \text{MZ}[b, m_b]$ . Putting  $T$  back together again we get Now

$$\begin{aligned} |r(T)| &= |r(S_a)| + |r(S_b)| + \sum_{i=1}^q |r(S_i)| + \sum_{\{i,j\} \subseteq \{a,1,\dots,q,b\}} |\{xy|z : x, y \in \mathcal{L}(S_i), z \in \mathcal{L}(S_j)\}| \\ &= |r(S_a)| + |r(S_b)| + \sum_{i=1}^q |r(S_i)| + \rho(m_a, m_1, \dots, m_q, m_b) \\ &\leq \text{MZ}[a, m_a] + \text{MZ}[b, m_b] + \sum_{i=1}^q \text{MZ}[i, m_i] + \rho(m_a, m_1, \dots, m_q, m_b) \\ &\leq \text{MANTAT}(a, b)[m] \end{aligned}$$

We now show that when  $\text{MANTAT}(a, b)[m] \geq 0$  there is a tree  $T \in \mathcal{T}(a, b)$  with  $m$  leaves such that  $|r(T)| = \text{MANTAT}(a, b)[m]$ . Choose the clique  $S$  of  $G$  and tuple  $m_a, m_1, \dots, m_q, m_b$  that last gave an update to  $M[m]$ . Choose  $x$  and  $y$  that give the maximum values  $\text{MZ}[a, m_a]$  and  $\text{MZ}[b, m_b]$ . For each  $s_i \in S$  choose a leaf  $s'_i$  giving the maximum value  $\text{MZ}[i, m_i]$ . By the induction step there are trees  $S_a, S_b, S_1, S_2, \dots, S_q$  such that  $|\mathcal{L}(S_a)| = m_a, |\mathcal{L}(S_b)| = m_b, |\mathcal{L}(S_i)| = m_i$  and  $|r(S_a)| = \text{MZ}[a, m_a], |r(S_b)| = \text{MZ}[b, m_b], |r(S_i)| = \text{MZ}[i, m_i]$  for  $i = 1, 2, \dots, q$ . Create a new root  $r$  and connect  $r$  to the roots of  $S_a, S_1, \dots, S_q, S_b$  giving the tree required.  $\square$

**Theorem 6.14** *The number of rooted triples in a MANTAT for  $\mathcal{T}$  can be calculated in  $O(kn^3 + dn^{2d})$  time.*

*Proof*

It takes  $O(kn^3)$  time to construct the sets  $R$  and  $F$ . We call the procedure  $\text{MANTAT}(a, b)$  for every pair of leaves  $a, b$ , storing the values as we proceed. We store the values calculated by  $\text{MANTAT}(a, b)$  in a table, so each need only be calculated once. The maximum value is then the maximum of  $\text{MANTAT}(a, b)[m]$  over all pairs of leaves  $a, b$  and values for  $m : 1 \leq m \leq n$ .

Looking at the algorithm for  $\text{MANTAT}(a, b)$  an iteration of the inner loop (steps 17 to 20) takes  $O(d)$  time, so the loop in steps 16 to 21 takes  $O(dn^d)$  time. The graph  $G$  has fewer than  $n$  vertices, so there are at most  $n^{d-2}$  cliques and the whole loop in steps 15 to 22 takes  $O(dn^d n^{d-2}) = O(dn^{2d-2})$  time. Steps 1 to 14 take only  $O(\min\{n^3, n^d\})$  time, so the entire procedure takes at most  $O(dn^{2d-2})$  time.

The procedure  $\text{MANTAT}(a, b)$  is called  $O(n^2)$  times, once for each pair of leaves. Hence calculating  $\text{MANTAT}(a, b)$  for all  $a, b$  takes  $O(dn^{2d})$  time, giving  $O(kn^3 + dn^{2d})$  for the whole problem.  $\square$

### 6.5.6 Maximum Number of Quartets Agreement Subtree

The unrooted equivalent of MANTAT involves not rooted triples but quartets:

INSTANCE: Profile  $\mathcal{T} = \{T_1, \dots, T_k\}$  of unrooted trees on leaf set  $L$ .

PROBLEM: Find an agreement subtree  $T$  for  $\mathcal{T}$  that contains a maximum number of quartets.

The algorithm solving this problem is essentially the same as the MANTAT algorithm, but a lot messier, so we omit it. For MANTAT we calculated the weight of a clique over all possible distributions of the number of leaves. In this unrooted algorithm we have to calculate the weight of a clique over all distributions of the number of leaves AND all distributions of the number of rooted triples. The resulting algorithm has complexity  $O(kn^3 + dn^{5d})$  for each different rooting, and  $O(kn^4 + dn^{5d+1})$  for the entire problem. We anticipate that a faster version is possible, but will be content with breaking the polynomial time barrier.

# Bibliography

- [1] E.N. Adams. Consensus techniques and the comparison of taxonomic trees. *Syst. Zool.*, 21:390–397, 1972.
- [2] E.N. Adams. N-trees as nestings: complexity, similarity, and consensus. *Journal of Classification*, 3:299–317, 1986.
- [3] R. Agarwala, V. Bafna, M. Farach, B. Narayanan, M. Paterson, and M. Thorup. On the approximability of numerical taxonomy (fitting distances by tree metrics). In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 365–372, 1996.
- [4] R. Agarwala and D. Fernández-Baca. A polynomial-time algorithm for the perfect phylogeny problem when the number of character states is fixed. *SIAM Journal on Computing*, 23(6):1216–1224, 1993.
- [5] A.V. Aho, J.E. Hopcroft, and J.D. Ullman. *The Design and Analysis of Computer Algorithms*. Addison-Wesley, Reading, MA, 1974.
- [6] A.V. Aho, T.G. Sagiv, T.G. Szymanski, and J.D. Ullman. Inferring a tree from lowest common ancestors with an application to the optimization of relational expressions. *SIAM Journal of Computing*, 10(3):405–421, 1981.
- [7] A. Amir and D. Keselman. Maximum agreement subtree in a set of evolutionary trees - metrics and efficient algorithms. In *35th Annual Symposium on Foundations of Computer Science*, pages 758–769, 1994.
- [8] K.J. Arrow. *Social Choice and Individual Values*. Wiley, New York, 1951.
- [9] H-J Bandelt and A. Dress. Reconstructing the shape of a tree from observed dissimilarity data. *Advances in Applied Mathematics*, 7:309–343, 1986.

- [10] H-J Bandelt and A.W.M Dress. Weak hierarchies associated with similarity measures - an additive clustering technique. *Bulletin of Mathematical Biology*, 51(1):133–166, 1989.
- [11] H-J Bandelt and A.W.M Dress. A canonical decomposition theory for metrics on a finite set. *Advances in Mathematics*, 92:47–105, 1992.
- [12] H.-J. Bandelt and A.W.M. Dress. A relational approach to split decomposition. Technical report, Universität Bielefeld, 1994.
- [13] H.-J. Bandelt and M. Steel. Symmetric matrices representable by weighted trees over a cancellative abelian monoid. *SIAM journal of Discrete Mathematics*, 8(4):517–525, 1995.
- [14] J. P. Barthélemy. Thresholded consensus for  $n$ -trees. *Journal of Classification*, 5(229-236), 1988.
- [15] J. P. Barthélemy, B. Leclerc, and B. Monjardet. On the use of ordered sets in problems of comparison and consensus of classifications. *Journal of Classification*, 3:187–224, 1986.
- [16] J. P. Barthélemy and F. R. McMorris. The median procedure for  $n$ -trees. *Journal of Classification*, 3:329–334, 1986.
- [17] J. P. Barthélemy, F. R. McMorris, and R. C. Powers. Independence conditions for consensus  $n$ -trees revisited. *Appl. Math. Lett.*, 4(5):43–46, 1991.
- [18] J. P. Barthélemy, F. R. McMorris, and R. C. Powers. Dictatorial consensus functions on  $n$ -trees. *Mathematical Social Sciences*, 25:59–64, 1992.
- [19] J.P. Barthélemy, F.R. McMorris, and R.C. Powers. Stability conditions for consensus functions defined on  $n$ -trees. *Mathematical Computer Modelling*, 22:79–87, 1995.
- [20] C. Benham, S. Kannan, and T. Warnow. Of chicken teeth and mouse eyes, or generalized character compatibility. In Z. Galil and U. Esko, editors, *Combinatorial Pattern Matching*, pages 17–26. Springer, 1995.

- [21] G. Birkoff and S. MacLane. *Algebra*. Macmillan Publishing Co., Inc., New York, 2nd edition edition, 1979.
- [22] H.L. Bodlaender, M.R. Fellows, and T.J. Warnow. Two strikes against perfect phylogeny. In *procs. of the 19th International Congress on Automata, Languages and Programming (ICALP)*, Springer-Verlag Lecture Notes in Computer Science, pages 273–287. 1992.
- [23] M. Bonet, C. Phillips, T.J. Warnow, and S. Yooseph. Construction evolutionary trees in the presence of polymorphic characters. Manuscript, 1995.
- [24] K. Bremer. Combinable component consensus. *Cladistics*, 6:369–372, 1990.
- [25] D. Bryant. Hunting for trees in binary character sets: efficient algorithms for extraction, enumeration and optimization. *Journal of Computational Biology*, 3(2):275–288, 1996.
- [26] D. Bryant and M. Steel. Extension operations on sets of leaf-labelled trees. *Advances in Applied Mathematics*, 16:425–453, 1995.
- [27] D. Bryant, M. Steel, and T. Warnow. Consensus techniques: a review and some new results. In the planning stage.
- [28] P. Buneman. The recovery of trees from measures of dissimilarity. In F.R. Hodson, D.G. Kendall, and P. Tautu, editors, *Mathematics in the Archaeological and Historical Sciences*, pages 387–395. Edinburgh University Press, Edinburgh, 1971.
- [29] P. Buneman. A characterisation of rigid circuit graphs. *Discrete Mathematics*, 9:205–212, 1974.
- [30] L. Cavalli-Sforza and A. Edwards. Phylogenetic analysis models and estimation procedures. *Evolution*, 32:550–570, 1967.
- [31] N. Christofides. *Graph Theory, an algorithmic approach*. Academic Press, London, 1975.
- [32] H. Colonius and H.H. Schulze. Tree structures for proximity data. *British Journal of Mathematical and Statistical Psychology*, 34:167–180, 1981.

- [33] M. Constantinescu and D. Sankoff. An efficient algorithm for supertrees. *Journal of Classification*, 12(1):101–112, 1995.
- [34] S.A. Cook. The complexity of theorem-proving procedures. In *Proc. 3rd Ann. ACM Symposium on Theory of Computing*, pages 151–158, New York, 1971.
- [35] G. Cucumel and Lapointe F.J. The average consensus procedure for weighted trees. Manuscript.
- [36] W. H. E. Day, F. R. McMorris, and Meronk D. B. Axioms for consensus functions based on lower bounds in posets. *Mathematical Social Sciences*, 12:185–190, 1986.
- [37] W.H.E. Day. Computational complexity of inferring phylogenies from dissimilarity matrices. *Bulletin of Mathematical Biology*, 49(4):461–467, 1987.
- [38] W.H.E. Day, D.S. Johnson, and D. Sankoff. The computational complexity of inferring rooted phylogenies by parsimony. *Mathematical Biosciences*, 81:33–42, 1986.
- [39] W.H.E. Day and D. Sankoff. Computational complexity of inferring phylogenies by compatibility. *Systematic Zoology*, 35(2):224–229, 1986.
- [40] M.C.H. Dekker. Reconstruction methods for derivation trees. Master's thesis, Department of Mathematics and Computer Science, Vrije Universiteit Amsterdam, 1986.
- [41] R.G. Downey and M.R. Fellows. Fixed-parameter tractability and completeness i: Basic results. *SIAM Journal of Computing*, 24:873–921, 1995.
- [42] A. Dress. Convex tree realizations of partitions. *Applied Mathematics Letters*, 5(3):3–6, 1992.
- [43] A. Dress, V. Moulton, and W. Terhalle. T-theory. Manuscript, 1995.
- [44] N. Eldredge and J. Cracraft. *Phylogenetic Patterns and the Evolutionary Process*. Columbia University Press, New York, 1980.

- [45] G.F. Estabrook, Johnson(jr) C.S., and F.R. McMorris. An idealized concept of the true cladistic character. *Mathematical Biosciences*, 23:263–272, 1975.
- [46] G.F. Estabrook, C.S. jr Johnson, and F.R. McMorris. A mathematical foundation of the analysis of cladistic character compatibility. *Mathematical Biosciences*, 29:181–187, 1976.
- [47] G.F. Estabrook and F.R. McMorris. When is one estimate of evolutionary relationships a refinement of another? *Journal of Mathematical Biology*, 10:367–73, 1980.
- [48] D.P. Faith. Distance methods and the approximation of most-parsimonious trees. *Systematic Zoology*, 34(3):312–325, 1985.
- [49] M. Farach, S. Kannan, and T. Warnow. A robust model for finding optimal evolutionary trees. *Algorithmica*, 13:155–179, 1995.
- [50] M. Farach, T. Przytycka, and M. Thorup. On the agreement of many trees. *Information Processing Letters*, 55:297–301, 1995.
- [51] J. Farris. Estimating phylogenetic trees from distance matrices. *The American Naturalist*, 106(951):645–667, 1972.
- [52] D. Fernández-Baca and J. Lagergren. A polynomial time algorithm for near perfect phylogeny. In *International Colloquium on Automata, Languages and Programming (ICALP)*, pages 670–680, 1996.
- [53] C.R. Finden and A.D. Gordon. Obtaining common pruned trees. *Journal of Classification*, 2:255–276, 1985.
- [54] W.M. Fitch and E. Margoliash. Construction of phylogenetic trees. *Science*, 155:276–284, 1967.
- [55] M.R. Garey and D.S. Johnson. *Computers and Intractability - A guide to the theory of NP-completeness*. W.H. Freeman and Company, 1979.
- [56] O. Gascuel. A reduction algorithm for approximating a (nonmetric) dissimilarity by a tree distance. *Journal of Classification*, 13(1):129–156, 1996.

- [57] W. Goddard, E. Kubicka, G. Kubicki, and F.R. McMorris. The agreement metric for labelled binary trees. *Mathematical Biosciences*, 123:215–226, 1994.
- [58] R.L. Graham and L.R. Foulds. Unlikelihood that minimal phylogenies for a realistic biological study can be constructed in reasonable computational time. *Mathematical Biosciences*, 60:133–142, 1982.
- [59] G.R. Grimwood. The Euclidean Steiner Tree Problem: simulated annealing and other heuristics. Master's thesis, Institute of Statistics and Operations Research, Victoria University of Wellington, New Zealand, 1994.
- [60] D. Gusfield. Efficient algorithms for inferring evolutionary trees. *Networks*, 21:19–28, 1991.
- [61] A.M. Hamel and M. Steel. Finding a maximum compatible tree is NP-hard for sequences and trees. *Applied Mathematics Letters*, 9(2):55–59, 1995.
- [62] S.B. Hedges, S. Kumar, and K. Tamura. Human origins and analysis of mitochondrial DNA sequences. *Science*, 255:737–739, 1991.
- [63] J. Hein, T. Jiang, L. Wang, and K. Zhang. On the complexity of comparing evolutionary trees. In Z. Galil and E. Ukkonen, editors, *Combinatorial Pattern Matching*, pages 177–190. Springer, 1995.
- [64] M.D. Hendy, C.H.C. Little, and D. Penny. Comparing trees with pendant vertices labelled. *SIAM Journal of Applied Mathematics*, 44(5):1054–1065, 1984.
- [65] M.D. Hendy and D. Penny. Spectral analysis of phylogenetic data. *Journal of Classification*, 10:5–24, 1993.
- [66] M.R. Henzinger, V. King, and T. Warnow. Constructing a tree from homeomorphic subtrees, with applications to computational evolutionary biology. In *Proceedings of the Seventh Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 333–340, 1996.
- [67] S. Kannan and T. Warnow. A fast algorithm for the computation and enumeration of perfect phylogenies. In *Proceedings of the Sixth Annual ACM-SIAM Symposium on Discrete Algorithms (SODA)*, pages 595–603, 1995.



- [68] S. Kannan, T. Warnow, and S. Yooseph. Computing the local consensus of trees. In *ACM-SIAM Symposium on Discrete Algorithms*, 1995.
- [69] S.K. Kannan and T.J. Warnow. Inferring evolutionary history from DNA sequences. *SIAM Journal of Computing*, 23(4):713–737, 1994.
- [70] S.K. Kannan and T.J. Warnow. Tree reconstruction from partial orders. *SIAM Journal of Computing*, 24(3):511–519, 1995.
- [71] E. Kubicka, G. Kubicki, and F. R. McMorris. An algorithm to find agreement subtrees. *Journal of Classification*, 12:91–100, 1995.
- [72] E. Kubicka, G. Kubicki, and F.R. McMorris. On agreement subtrees of two binary trees. In *Congressus Numerantium 88*, pages 217–224, 1992.
- [73] W.J. Le Quesne. A method of selection of characters in numerical taxonomy. *Systematic Zoology*, 18:201–205, 1969.
- [74] D.R. Maddison, M. Ruvolo, and D.L. Swofford. Geographic origins of human mitochondrial DNA: phylogenetic evidence from control region sequences. *Systematic Biology*, 41(1):111–124, 1992.
- [75] T. Margush and F. R. McMorris. Consensus  $n$ -trees. *Bulletin of Mathematical Biology*, 43(2):239–244, 1981.
- [76] F. R. McMorris, D. B. Meronk, and D. A. Neumann. A view of some consensus methods for trees. In J. Felsenstein, editor, *Numerical Taxonomy*, pages 122–125. Springer-Verlag, 1983.
- [77] F. R. McMorris and R. C. Powers. Consensus weak hierarchies. *Bulletin of Mathematical Biology*, 53:679–684, 1991.
- [78] F. R. McMorris and R. C. Powers. Consensus functions on trees that satisfy an independence axiom. *Discrete Applied Mathematics*, 47:47–53, 1993.
- [79] F. R. McMorris and M. A. Steel. The complexity of the median procedure for binary trees. In *Proceedings of the International Federation of Classification Societies, 1993*, New York, 1994.

- [80] F.R. McMorris. On the compatibility of binary qualitative taxonomic characters. *Bulletin of Mathematical Biology*, 39:133–138, 1977.
- [81] F.R. McMorris. Axioms for consensus functions of undirected phylogenetic trees. *Mathematical Biosciences*, 74:17–21, 1985.
- [82] F.R. McMorris, T. Warnow, and T. Wimer. Triangulating vertex-colored graphs. *SIAM Journal of Discrete Mathematics*, 7(2):296–306, 1994.
- [83] C.A. Meacham and G.F. Estabrook. Compatibility methods in systematics. *Annual Review of Ecology and Systematics*, 16:431–446, 1985.
- [84] M. Nei. Relative efficiencies of different tree-making methods for molecular data. In M. M. Miyamoto and J. Cracraft, editors, *Phylogenetic analysis of DNA sequences*, pages 90–128. Oxford University Press, 1991.
- [85] G. Nelson. Cladistic analysis and synthesis: principles and definitions, with a historical note on Adanson's *Familles des Plantes (1763-1764)*. *Systematic Zoology*, 28:1–21, 1979.
- [86] D. A. Neumann. Faithful consensus methods for  $n$ -trees. *Mathematical Biosciences*, 63:271–287, 1983.
- [87] M.P. Ng, M. Steel, and N.C. Wormald. The difficulty of constructing a leaf-labelled tree including or avoiding given subtrees. Technical Report 130, Department of Mathematics and Statistics, University of Canterbury, 1995.
- [88] M.P. Ng and N.C. Wormald. Reconstruction of rooted trees from subtrees. *Discrete Applied Mathematics*, 69(1-2):19–31, 1996.
- [89] D. M. Page. Tracks and trees in the antipodes: A reply to Humphries and Seberg. *Systematic Zoology*, 39(3):288–299, 1990.
- [90] D. Penny, M. Steel, P.J. Waddell, and M.D. Hendy. Improved analysis of human mtDNA sequences support a recent African origin for *Homo sapiens*. *Molecular Biology and Evolution*, 12(5):863–882, 1995.

- [91] C. Phillips and T. J. Warnow. The asymmetric median tree - a new model for building consensus trees. In *Proceedings of the Combinatorial Matching Conference, Laguna Beach, CA*, 1996.
- [92] D. Robinson. Comparison of labelled trees with valency three. *Journal of Combinatorial Theory*, 11(2):105–119, 1971.
- [93] D.F. Robinson and L.R. Foulds. Comparison of phylogenetic trees. *Mathematical Biosciences*, 53:131–147, 1981.
- [94] A. Rzhetsky and M. Nei. A simple method for estimating and testing minimum-evolution trees. *Molecular Biology and Evolution*, 9(5):945–967, 1992.
- [95] A. Rzhetsky and M. Nei. Theoretical foundation of the minimum-evolution method of phylogenetic inference. *Molecular Biology and Evolution*, 10(5):1073–1095, 1993.
- [96] N. Saitou and M. Nei. The neighbor-joining method: a new method for reconstructing phylogenetic trees. *Molecular Biology and Evolution*, 24:189–204, 1987.
- [97] S. Sattath and A. Tversky. Additive similarity trees. *Psychometrika*, 42(3):319–345, 1977.
- [98] E. Schröder. Vier combinatorische probleme. *Zeitschrift für Mathematische Physik*, 15:361–376, 1870.
- [99] J.M.S. Simões Pereira. A note on the tree realizability of a distance matrix. *Journal of Combinatorial Theory*, 6:303–310, 1969.
- [100] P.H.A. Sneath and R.R. Sokal. *Numerical Taxonomy*. W.H. Freeman, San Francisco, 1973.
- [101] M. Steel. The complexity of reconstructing trees from qualitative characters and subtrees. *Journal of Classification*, 9:91–116, 1992.
- [102] M. Steel and T. Warnow. Kaikoura tree theorems: Computing the maximum agreement subtree. *Information Processing Letters*, 48(2):77–82, 1993.

- [103] M.A. Steel and D. Penny. Distribution of tree comparison metrics - some new results. *Systematic Biology*, 42(2):126–141, 1993.
- [104] R. Stinebrickner. s-consensus trees and indices. *Bulletin of Mathematical Biology*, 46:923–935, 1984.
- [105] R. Stinebrickner. s-consensus index method: an additional axiom. *Journal of Classification*, 3:319–327, 1986.
- [106] D. Swofford, G.J. Olsen, P.J. Waddell, and D.M. Hillis. Phylogenetic inference. In D.M. Hillis, C. Moritz, and B.K. Mable, editors, *Molecular Systematics*, pages 407–514. Sinauer, 2nd edition, 1996.
- [107] D. L. Swofford. When are phylogeny estimates from molecular and morphological data incongruent? In M. M. Miyamoto and J. Cracraft, editors, *Phylogenetic analysis of DNA sequences*, pages 295–333. Oxford University Press, 1991.
- [108] A.R. Templeton. Human origins and analysis of mitochondrial dna sequences. *Science*, 255:737–737, 1991.
- [109] W. Vach. Least squares approximation of additive trees. In *Conceptual and Numerical Analysis of Data*, pages 230–238, 1989.
- [110] W. Vach and P.O. Degens. Least squares approximation of additive trees to dissimilarities-characterizations and algorithms. *Computational Statistics Quarterly*, 3:203–218, 1991.
- [111] L. Vigilant, M. Stoneking, H. Harpending, K. Hawkes, and A.C. Wilson. African populations and the evolution of human mitochondrial dna. *Science*, 253:1503–1507, 1991.
- [112] T. Warnow. Constructing phylogenetic trees efficiently using compatibility criteria. *New Zealand Journal of Botany*, 31:239–248, 1993.
- [113] T. Warnow. Tree compatibility and inferring evolutionary history. *Journal of Algorithms*, 16:388–407, 1994.
- [114] D.J.A. Welsh. *Matroid Theory*. Academic Press, London, 1976.

- [115] E.O. Wiley, D. Siegel-Causey, F.R. Brooks, and V.A. Funk. *The Compleat Cladist*. Museum of Natural History, University of Kansas, Lawrence, Kansas, 1991.
- [116] M. Wilkinson. Common cladistic information and its consensus representation: reduced adams and reduced cladistic consensus trees and profiles. *Systematic Biology*, 43(3):343–368, 1994.



# Appendix A

## A List of NP-complete problems

### A.1 General Compatibility Problems

#### QUARTET COMPATIBILITY

INSTANCE: A set  $Q$  of quartets on a leaf set  $L$ .

QUESTION: Is  $Q$  compatible? That is, is there a tree  $T$  such that  $Q \subseteq q(T)$ ?

*Reference:* Steel [101]. Transformation from BETWEENNESS.

*Comment:* Can be solved in polynomial time if there is a leaf common to all quartets. Remains NP-complete, however, if there are two leaves  $\alpha, \beta$  such that each quartet in  $Q$  contains either  $\alpha$  or  $\beta$  [87].

#### SPLIT-QUARTET COMPATIBILITY.

INSTANCE: Set  $Q$  of quartets on a leaf set  $L$ .

QUESTION: Is there are non-trivial split not contradicted by  $Q$ ? That is, is there a split  $A|B$  of  $L$  such that there is no quartet  $ab|cd \in Q$  with  $a, c \in A$  and  $b, d \in B$ ?

*Reference:* Chapter 2, section 2.4.2. Transformation from 3SAT.

#### TREE COMPATIBLE WITH PARTIAL ORDER

INSTANCE: Leaf set  $L$ . Partial order  $\leq_P$  on pairwise distances in  $L$ .

QUESTION: Is there a weighted tree  $T$  such that the leaf to leaf distance metric in  $T$  satisfies the partial order  $\leq_P$ ?

*Reference:* Kannan and Warnow [70]. Transformation from QUARTET COMPATIBILITY.

*Comment:* Remains NP-complete if the tree  $T$  is forced to have unit edge weights, and if the partial order is defined only on subsets of three leaves (TOM and POM consistency in [70]).

### RESOLVED TREE COMPATIBLE WITH TRIPLES AND FANS

INSTANCE Set  $R$  of rooted triples and set  $F$  of fans. Number  $K \geq 0$ .

QUESTION Is there a tree  $T$  with  $K$  or more internal edges such that  $R \subseteq r(T)$  and  $F \subseteq f(T)$ ?

*Reference:* Chapter 2, section 2.6.3. Transformation from GRAPH  $K$ -COLORABILITY.

*Comment:* Remains NP-complete even if  $R = \emptyset$  and there is a leaf common to all fans. (Chapter 2, Theorem 2.21). Can be solved in polynomial time if  $F = \emptyset$  (Compatibility of rooted triples).

## A.2 Maximum Compatible Subset Problems

### MAXIMUM COMPATIBLE SUBSET OF CHARACTERS

INSTANCE: Set  $C$  of  $r$ -state characters. Positive integer  $B \leq |C|$ .

QUESTION: Is there a compatible subset  $C' \subseteq C$  such that  $B \leq |C'|$ ?

*Reference:* Day and Sankoff [39]. Transformation from CLIQUE.

*Comment:* Remains NP-complete if  $r = 2$  (binary characters) [39]. Clearly it is also NP-complete if we assign weights to characters and look for a subset with a weight greater than  $B$ . Also NP-complete if the characters in  $C$  correspond to weakly compatible splits (Chapter 4, Theorem 4.12). Becomes polynomial if we insist that  $C' = \sigma(T)$  for some tree  $T$  with fixed maximum vertex degree  $d$  (Chapter 4).

### MAXIMUM COMPATIBLE SUBSET OF ROOTED TRIPLES

INSTANCE: Set  $R$  of rooted triples. Positive integer  $B \leq |R|$ .

QUESTION: Is there a compatible subset  $R' \subseteq R$  such that  $B \leq |R'|$ ?

*Reference:* Proof suggested by Warnow. See Chapter 2, section 2.6.1. Transformation from FEEDBACK ARC SET.

*Comment:* Remains NP-complete even if we insist that  $R' = r(T)$  for some binary tree  $T$  (transformation from FORBIDDEN TRIPLES).



**MAXIMUM COMPATIBLE SUBSET OF QUARTETS**

INSTANCE: Set  $Q$  of quartets. Positive integer  $B \leq |Q|$ .

QUESTION: Is there a compatible subset  $Q' \subset Q$  such that  $B \leq |Q'|$ ?

*Reference:* Trivial transformation from MAXIMUM COMPATIBLE SUBSET OF ROOTED TRIPLES, or from QUARTET COMPATIBILITY.

**SUBCHARACTER COMPATIBILITY**

INSTANCE: A collection  $S$  of splits of a set  $L$ , integer  $k$ .

QUESTION: Is there a subset  $L'$  of  $L$  of size at least  $k$  such that the set of splits  $S|_{L'} = \{A \cap L' | B \cap L' : A|B \in S\}$  is compatible?

*Reference:* Hamel and Steel [61].

**A.3 Forbidden Subtrees Problems****FORBIDDEN QUARTETS**

INSTANCE: A collection  $Q$  of quartets on leaf set  $L$ .

QUESTION: Is there a binary tree  $T$  on  $L$  such that  $q(T) \cap Q = \emptyset$ ?

*Reference:* Trivial transformation from FORBIDDEN TRIPLES (below)

**FORBIDDEN SUBTREES**

INSTANCE: A collection  $S$  of rooted binary trees whose leaf sets are subsets of a label set  $L$ .

QUESTION: Is there a leaf-labelled rooted binary tree  $T$  with label set  $L$  having no induced subtree  $T|_{L'}$  equal to a tree in  $S$ ?

*Reference:* Ng, Steel and Wormald [87]. Transformation from BETWEENNESS.

*Comment:* Remains NP-complete if all the trees in  $S$  have three leaves (FORBIDDEN TRIPLES below).

**FORBIDDEN TRIPLES**

INSTANCE: A collection  $R$  of rooted triples on leaf set  $L$ .

QUESTION: Is there a rooted binary tree  $T$  such that  $R \cap r(T) = \emptyset$ ?

*Reference:* Chapter 2, section 2.6.2. Transformation from 3SAT.

*Comment:* Solvable in polynomial time if we constrain  $T$  to be a caterpillar tree. (Chapter 2)

## A.4 Consensus Tree Problems

### BINARY MEDIAN TREE

INSTANCE: A profile  $\mathcal{T} = \{T_1, T_2, \dots, T_k\}$  of trees on a leaf set  $L$ . Number  $K$ .

QUESTION: Is there a binary tree  $T$  such that

$$\sum_i d(T, T_i) \leq K$$

where  $d(T, T_i) = |\beta(T) - \beta(T_i)| + |\beta(T_i) - \beta(T)|$ ?

*Reference:* McMorris and Steel [79]. Transformation from 3-DIM MATCHING

*Comment:* Solvable in polynomial time if we drop the constraint that  $T$  be binary [16].

### MEDIAN SUPERTREE

INSTANCE: A collection  $\mathcal{T} = \{T_1, \dots, T_k\}$  of rooted trees whose leaf sets are subsets of a label set  $L$ . Number  $K$ .

QUESTION: Is there a rooted tree  $T$  such that

$$\sum_i d(T|_{\mathcal{L}(T_i)}, T_i) \leq K$$

where  $d(T, T') = |\sigma(T) - \sigma(T')| + |\sigma(T') - \sigma(T)|$ ?

*Reference:* Transformation from MAXIMUM COMPATIBLE SUBSET OF ROOTED

TRIPLES: Put  $\mathcal{T} = R$ , then  $d(T|_{\mathcal{L}(T_i)}, ab|c) = 0$  if  $ab|c \in r(T)$ , otherwise  $d(T|_{\mathcal{L}(T_i)}, ab|c) = 2$ . Hence  $T$  is a median supertree if and only if  $r(T) \cap R$  is a maximum compatible subset of rooted triples.

### MAXIMUM AGREEMENT SUBTREE

INSTANCE: Profile  $\mathcal{T} = \{T_1, \dots, T_k\}$  of trees on leaf set  $L$ . Number  $K$ .

QUESTION: Is there a subset  $L' \subseteq L$  such that  $T_1|_{L'} = T_2|_{L'} = \dots = T_k|_{L'}$  and  $K \leq |L'|$ ?

*Reference:* Amir and Keselman [7]. Transformation from 3-DIM MATCHING.

*Comment:* Shown to be NP-complete for three trees [7], solvable in polynomial time for two trees [102, 57]. Can be solved in polynomial time for fixed  $d$  if one of the trees in  $\mathcal{T}$  has maximum vertex degree  $d$  [7, 50].

#### MAXIMUM COMPATIBLE SUBTREE

INSTANCE: Profile  $\mathcal{T} = \{T_1, \dots, T_k\}$  of trees on leaf set  $L$ . Number  $K$ .

QUESTION: Is there a tree  $T$  and a subset  $L' \subseteq L$  such that  $T_{i|L'} \preceq T$  for all  $i = 1, 2, \dots, k$  and  $K \leq |L'|$ ?

*Reference:* Two trees: [63]. NP-complete for six trees by [61].

#### ASYMMETRIC MEDIAN TREE

INSTANCE: A profile  $\mathcal{T} = \{T_1, T_2, \dots, T_k\}$  of trees on a leaf set  $L$ . Number  $K$ .

QUESTION: Is there a tree  $T$  such that

$$\sum_i d(T, T_i) \leq K$$

where  $d(T, T_i) = |\beta(T_i) - \beta(T)|$ ?

*Reference:* Shown to be NP-complete for three trees by [91]. Transformation from 3-DIM MATCHING

## A.5 Perfect Phylogeny and Tandifications

#### PERFECT PHYLOGENY

INSTANCE: A collection  $C$  of qualitative characters on leaf set  $L$ .

QUESTION: Is there a perfect phylogeny for  $C$ ? A tree  $T$  is a perfect phylogeny for  $C$  if for every  $c \in C$  the tree  $T$  can be broken down into subtrees  $T_1, \dots, T_s$  so that two leaves are in the same subtree if and only if they are assigned the same state by  $c$ .

*Reference:* Proved independently by Steel [101] (transformation from QUARTET COMPATIBILITY) and Bodlaender, Fellows and Warnow [22] (transformation from 3SAT via TRIANGULATING COLORED GRAPHS).

*Comment:* Can be solved in polynomial time when  $k = |C|$  is fixed [82] or when

$n = |L|$  is fixed (by searching all trees) or when the number of states of characters in  $C$  is bounded [4, 68]. See also [52] for polynomial time algorithms for “near-perfect” phylogeny with bounded number of states. Extensions to generalised or polymorphic characters have also been shown to be NP-complete, even with bounds on the number of states [20, 23].

## A.6 Parsimony

Given a rooted or unrooted tree  $T$  with leaf set  $L$  and a character  $c$  on  $L$  an extension  $\chi_c$  of  $c$  on  $T$  is a function from the vertices of  $T$  to the states of  $c$  that agrees with  $c$  on  $L$ . The length  $l(\chi_c)$  of an extension on  $T$  is the number of edges in  $T$  which are assigned different states by  $\chi_c$ .

### PARSIMONY

INSTANCE: Set of characters  $C$  on leaf set  $L$ . Number  $B$ .

QUESTION: Is there a tree  $T$  on leaf set  $L$  and an extension  $\chi_c$  on  $T$  for each  $c \in C$  such that

$$\sum_{c \in C} l(\chi_c) \leq B.$$

*Reference:* Shown NP-complete by [58, 38]. *Comment:* Remains NP-complete for binary characters and for ordered or unordered characters. The problem with no constraints on the extensions  $\chi_c$  is called parsimony with the Fitch criterion. Alternatives are (i) Wagner parsimony, where the characters are ordered. (ii) Camin-Sokal parsimony for rooted trees, where  $\chi_c$  the state of a vertex  $v$  assigned by  $\chi_c$  must be equal to or a derivative of the state assigned to any ancestor of  $v$ . (iv) Dollo parsimony, for rooted trees, where character state change is reversible but the origin in  $T$  for every character state is unique. (v) Generalised, where the formula for calculating the length of an extension is expanded to include differently weighted transitions. Each alternative is NP-complete.

The problem becomes polynomial [52] if we fix  $r$  and require that the imperfection of  $T$  is bounded, where the imperfection equals

$$\sum_{c \in C} (l(\chi_c) - (\text{number of states of } c)).$$

## A.7 Distance Based Methods

### FITTING ADDITIVE TREES

INSTANCE Metric  $d$  on leaf set  $L$ . Number  $K$ .

QUESTION Is there an additive metric  $p$  such that

$$\|p - d\|_\alpha \leq K?$$

*Reference:* For  $\alpha = 1, 2$ . [36] Transformation from FITTING ULTRAMETRIC TREES.

*Comment:* Remains NP-complete if we specify that  $p$  is ultrametric, though there is a polynomial time algorithm for when  $p$  is constrained to be ultrametric and  $\alpha = \infty$ . NP-complete for additive trees when  $\alpha = \infty$  [3].

### MATRIX COMPLETION TO ADDITIVE METRIC

INSTANCE Matrix  $M$  for which only some entries are known. An unset matrix entry is indicated by  $M[i, j] = "*"$ .

QUESTION Does a matrix  $M'$  corresponding to an additive metric exist such that  $M'[i, j] = M[i, j]$  whenever  $M[i, j] \neq "*"$ .

*Reference:* Farach, Kannan and Warnow [49]. Transformation from QUARTET COMPATIBILITY.

*Comment:* Solvable in polynomial time if we specify that  $M'$  corresponds to an ultrametric distance metric [49].

## A.8 Open Problems

### QUARTETS DEFINING TREE

INSTANCE: Compatible set  $Q$  of quartets. Binary tree  $T \in \langle Q \rangle$ .

QUESTION: Does  $\langle Q \rangle = \{T\}$ ?

### HUNTING FOR TREE IN $r$ -STATE CHARACTERS

INSTANCE: Set of  $r$ -state characters  $C$  on leaf set  $L$ .

QUESTION: Is there a subset  $C' \subseteq C$  such that there is exactly one binary tree  $T$

compatible with the characters  $C'$ ?

# Appendix B

## List of Algorithms

	<i>Name</i>	<i>Function</i>	<i>Page</i>
1	CONSTRUCTTREE	Constructs trees compatible with a set of quartets	34
2	ONETREE	Constructs a tree compatible with a set of rooted triples	36
3	CREATETABLE	Constructs decomposition table	80
4	COUNTTREES	Counts $d$ -trees with clusters in a given set.	81
5	GETSUBCOLLECTIONS	Extracts trees from a decomposition table	84
6	MAXWEIGHTTREE	Calculates weight of maximum weight $d$ -tree with clusters in a given set.	
7	MINIMUM $d$	Calculates smallest $d$ such that a given set of clusters $C$ contains the clusters of some $d$ -tree.	88
8	INTERSECTD	Calculates the intersection of two decomposition tables.	92
9	PRUNETOINCLUDE	Removes rows and tuples from a decomposition table to exclude trees not containing a given cluster.	94
10	COUNTINCLUDETREE	Counts trees in a decomposition table that include a given cluster	95
11	MAXINCLUDETREE	Calculates maximum weight of trees in a decomposition table that include a given cluster.	97
12	QUARTETSPLITS	Constructs the set of splits $A B$ such that $q(A B)$ is contained in a given quartet set $Q$	106
13	BUILDTREE	Constructs a decomposition table containing all binary trees in $\langle Q \rangle$ , where $Q$ is a given set of quartets	108
14	CALCULATE $Ad$	Given a tree $T$ , calculates $A^T d$ where $d$ is a vector and $A$ is the topological matrix for $T$ .	123

	<i>Name</i>	<i>Function</i>	<i>Page</i>
15	CALCULATEEDGES	Optimal time algorithm for calculating edge lengths of a tree under OLS	139
16	CALCULATEME	Calculates the lowest ME score of any binary tree with splits in a given set	143
17	EXTRACTMETREE	Extracts the binary tree with lowest ME score that has splits in a given set	144
18	EXTRACTMESUBTREE	Recursive procedure called by EXTRACTMETREE	145
19	ADAMSTREE	Constructs the Adams consensus tree	158
20	AGSUBTREE	Returns an arbitrary agreement subtree of a given profile	177
21	MAST	Calculates size of Maximum Agreement Subtree	180
22	MIST	Calculates size of Maximum Information Agreement Subtree	183
23	MANTAT	Calculates the maximum number of rooted triples of any agreement subtree of a given profile	186



# Appendix C

## List of Symbols

<i>Symbol</i>	<i>Usage</i>	<i>Meaning</i>	<i>Page</i>
$\in$	$x \in S$	$x$ is an element of $S$	
$\notin$	$x \notin S$	$x$ is not an element of $S$	
$\subset$	$A \subset B$	$A$ is a <i>proper</i> subset of $B$	
$\subseteq$	$A \subseteq B$	$A$ is a subset of $B$	
$\emptyset$	$\emptyset$	The empty set	
$\{:\}$	$\{x : \dots\}$	The set of $x$ such that $\dots$	
$O(\cdot)$	$O(f(n))$	Order of $f(n)$	10
$P$	$P$	The set of polynomial time solvable problems.	10
$\mathcal{L}()$	$\mathcal{L}(T)$	The leaf set of tree $T$	6
$ab c$	$ab c$	A rooted triple	10
$ab cd$	$ab cd$	A quartet	9
$(a, b, c)$	$(a, b, c)$	A fan tree with leaves $a, b, c$	10
$A B$	$A B$	A split (bipartition) with blocks $A$ and $B$	8
$r()$	$r(T)$	The set of rooted triples of a rooted tree $T$	10
$f()$	$f(T)$	The set of fan triples induced by a rooted tree $T$	10
$q()$	$q(T)$	The set of quartets of an unrooted tree $T$	9
	$q(A B)$	The set $\{a_1a_2 b_1b_2 : a_1, a_2 \in A, b_1, b_2 \in B\}$	27
$\sigma()$	$\sigma(T)$	The set of clusters in a rooted tree $T$	8
$\beta()$	$\beta(T)$	The set of splits in an unrooted tree $T$	8
$\trianglelefteq$	$S \trianglelefteq T$	$T$ extends $S$	19
$\langle \rangle$	$\langle \mathcal{T} \rangle$	The span of the set of trees $\mathcal{T}$	19
$\prec_T$	$A \prec_T B$	$A$ nests in $B$	22

<i>Symbol</i>	<i>Usage</i>	<i>Meaning</i>	<i>Page</i>
$\text{ROOT}()$	$\text{ROOT}(T, x)$	The tree obtained by removing leaf $x$ from the unrooted tree $T$ and rooting $T$ at the adjacent vertex.	7
	$\text{ROOT}(A B, x)$	The set $A$ if $x \in B$ or the set $B$ if $x \in A$	90
	$\text{ROOT}(\mathcal{S}, x)$	The set $\{\text{ROOT}(A B, x) : A B \in \mathcal{S}\}$	90
$\text{UNROOT}()$	$\text{UNROOT}(T, x)$	The tree obtained by attaching $x$ to the root of $T$ and unrooting $T$ .	7
	$\text{UNROOT}(A, x)$	The split $A (L - A) \cup \{x\}$ where $A$ is a subset of $L$ .	90
	$\text{UNROOT}(\mathcal{C}, x)$	The set $\{\text{UNROOT}(A, x) : A \in \mathcal{C}\}$	90
$(\cdot)$	$(A)B$	$n$ -taxon statement	23
$\cdot $	$T _A$	The subtree of $T$ induced by $A$	18
	$\mathcal{C} _X$	The set $\{A \cap X : A \in \mathcal{C}, A \cap X \neq \emptyset\}$ , where $\mathcal{C}$ is a set of clusters	22
	$\mathcal{S} _X$	The set $\{A \cap X B \cap X : A B \in \mathcal{S}, A \cap X \neq \emptyset, B \cap X \neq \emptyset\}$	22
$\prec$	$\{a, b\} \prec \{c, d\}$	Least common ancestor of $a$ and $b$ is a descendent of the least common ancestor of $c$ and $d$	24
$\overline{\cdot}$	$\overline{Q}$	The closure of the set of quartets $Q$	52
	$\overline{R}$	The closure of the set of rooted triples $R$	52
$\vdash$	$Q \vdash ab cd$	Quartet inference rule	50
	$R \vdash ab c$	Rooted triple inference rule	52
$CS()$	$CS(Q)$	The set of splits $\{A B : a_1b_1 a_2b_2 \notin Q \text{ for all } a_1, a_2 \in A, b_1, b_2 \in B\}$	30
$[\cdot, \cdot]$	$[R, S]$	The rooted triple graph for $R$ on leaf set $S$	35
$\text{freq}()$	$\text{freq}(X, \mathcal{T})$	The number of trees in $\mathcal{T}$ containing the cluster $X$	168
	$\text{freq}(A B, \mathcal{T})$	The number of trees in $\mathcal{T}$ containing the splits $A B$	168
$\mathcal{T}$	$\mathcal{T}$	A set of trees	
	$\mathcal{T}(D)$	The set of trees that can be extracted from the decomposition table $D$	92
	$\mathcal{T}(C, d)$	The set of rooted $d$ -trees with clusters in $C$	92
	$\mathcal{T}(S, d)$	The set of unrooted $d$ -trees with splits in $S$	92
	$\mathcal{T}(a, b)$	The set of agreement subtrees of $\mathcal{T}$ with roots equal to the least common ancestor of $a$ and $b$	175